A MINI PROJECT REPORT ON

"Indexing for Hospital Management System"

Mini-Project Report submitted in Partial fulfilment of the requirement for the 6th Semester File Structures Laboratory with Mini-Project

[17ISL68]

Bachelor of Engineering
in
Information Science and Engineering

Submitted by
HARSHITHA.C [1JT17IS013]

Department of Information Science and Engineering
Jyothy Institute of Technology
Tataguni, Bengaluru-560082

Jyothy Institute of Technology
Tataguni, Bengaluru-560082
Department of Information Science and Engineering

CERTIFICATE

Certified that the mini project work entitled **“INDEXING”** carried out by **HARSHITHA.C [1JT17IS013]** bonafide student of Jyothy Institute of Technology, in partial fulfilment for the award of **Bachelor of Engineering** in **Information Science and Engineering** department of the **Visvesvaraya Technological University, Belagavi** during the year **2019-2020**. 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 in respect of Mini Project work prescribed for the said Degree.

Mr. Vadiraja A
Guide, Asst. Professor
Dept. Of ISE

Dr. Harshvardhan Tiwari
Associate Professor and HoD
Dept. Of ISE

External Viva Examiner

1.
2.

Signature with Date :

ACKNOWLEDGEMENT

Firstly, I am very grateful to this esteemed institution **"Jyothy Institute of Technology**" for providing me an opportunity to complete my project.

I express my sincere thanks to our Principal **Dr. Gopalakrishna K for** providing me with adequate facilities to undertake this project.

I would like to thank **Dr. Harshwardhan Tiwari, Professor and Head** of Information Science and Engineering Department for providing for his valuable support.

I would like to thank our guides **Mr. Vadiraja A, Asst. Prof.** for her keen interest and guidance in preparing this work.

Finally, I would thank all my friends who have helped me directly or indirectly in this project.

HARSHITHA.C [1JT17IS013]

ABSTRACT

The project title is “Indexing” which is used in the application and for distinguishing the type of file structure to be used.

The work provides a simple and attractive and it creates the simple index and simplify the work as well as it reduces the time taken and t takes the same time to search the record in the first and also the record in the last.

File structure is a combination of representations for data in files and of operations for accessing the data.
A file structure allows applications to read, write and modify data.

Indexing is the ordered file where data file is ordered on a key field. An index lets us impose order on a file without rearranging the file. It is a technique to efficiently retrieve records from files based on some attributes on which the indexing has been done.

Hospital management manages patients records. It is done by using the JAVA.
The hospital management system mainly uses file handling to perform basic operations like how to add, modify, search and delete record using file. It is very flexible to incorporate future modifications, modules, and features as per user requirements.

TABLE OF CONTENTS

CHAPTER 1
INTRODUCTION

INTRODUCTION

1.1 Introduction to FILE STRUCTURES:

File Structures is the Organization of Data in Secondary Storage Device in such a way that minimize the access time and the storage space. A File Structure is a combination of representations for data in files and of operations for accessing the data. A File Structure allows applications to read, write and modify data. It might also support finding the data that matches some search criteria or reading through the data in some particular order. Records are collected into logical units called files .They enable one to refer to a set of records by name ,the file name. The records within a file are often organised according to relationships between record .These logical organisation are known as FILE STRUCTURES.

The goal of File Structure is to get the information we need with one access to the disk. If it is not possible, then get the information with as few accesses as possible. Group information so that we are likely to get everything we need with only one trip of the disk. It is relatively easy to come up with File Structure designs that meet the general goals when the files never change. When files grow or shrink when information is added and deleted, it is much more difficult.

Early Work assumed that files were on tape. Access was sequential and the cost of access grew in direct proportion to the size of the file. As files grew very large, unaided sequential access was not a good solution. Disks allowed for direct access. Indexes made it possible to keep a list of keys and pointers in a small file that could be searched very quickly. With the key and pointer, the user had direct access to the large, primary file. As indexes also have a sequential flavour, when they grew too much, they also became difficult to manage. The idea of using tree structures to manage the index emerged in the early 60's. However, trees can grow very unevenly as records are added and deleted, resulting in long searches requiring many disk accesses to find a record. In 1963, researchers came up with the idea of AVL trees for data in memory.

AVL trees, however, did not apply to files because they work well when tree nodes are composed of single records rather than dozens or hundreds of them.

In the 1970's came the idea of B-Trees which require an O(logk N) access time where N is the number of entries in the file and kth number of entries indexed in a single block of the BTree structure --> B-Trees can guarantee that one can find one file entry among millions of others with only 3 or 4 trips to the disk.

1.2 Introduction to JAVA:

Java is a general-purpose programming language that is class-based, object-oriented, and designed to have as few implementation dependencies as possible

Like any programming language, Java language has its own structure, syntax rules, and programming paradigm. The Java language's programming paradigm is based on concept of OOP, which the language's features support.

The Java language is a C-language derivative, so its syntax rules look much like C's. For example, code blocks are modularized into methods and delimited by braces and variables are declared before they are used.

Structurally, Java language starts with packages. A packages is the Java language's namespace mechanism. Within packages are classes, and within classes are methods variables, constants, and more. You learn about the parts of the java language in this tutorial.

1.3 Introduction to INDEXING:

A structure containing a set of entries, each consisting of a key field and a reference field, which is used to locate records in a data file

Key Field: The part of an index which contains keys.

Reference Field: The part of an index which contains information to locate records.

- An index imposes order on a file without rearranging the file.

A Simple Index for Entry-Sequenced Files

An index in which the entries are a key ordered linear list

- Simple indexing can be useful when the entire index can be held in memory.
- Changes (additions and deletions) require both the index and the data file to be changed.
- Updates affect the index if the key field is changed, or if the record is moved.
- An update which moves a record can be handled as a deletion followed by an addition.

A file in which the record order is determined by the order in which they are entered

- The physical order of records in the file may not be the same as order of entry, because of record deletions and space reuse.
- The index should be read into memory when the data file is opened.
- Searching of a simple index on disk takes too much time.
- Maintaining a simple index on disk in sorted order takes too much time.

Secondary Index: An index built on a secondary key.

- Secondary indexes can be built on any field of the data file, or on combinations of fields.
- Secondary indexes will typically have multiple locations for a single key.
- Changes to the data may now affect multiple indexes.

- The reference field of a secondary index can be a direct reference to the location of the entry in the data file.
- The reference field of a secondary index can also be an indirect reference to the location of the entry in the data file, through the primary key.

Indexed Files Sequential search is even slower on disk/tape than in main memory. Try to improve performance using more sophisticated data structures. An index for a file is a list of key field values occurring in the file along with the address of the corresponding record in the mass storage. Typically the key field is much smaller than the entire record, so the index will fit in main memory. The index can be organized as a list, a search tree, a hash table, etc. To find a particular record: Search the index for the desired key. When the search returns the index entry, extract the record's address on mass storage. Access the mass storage at the given address to get the desired record. Multiple indexes, one per key field, allow searches based on different file.

CHAPTER 2
DESIGN

2.1 Domain understanding

The main object of this project is to perform operations like searching, deleting, inserting and indexing the records in the separate file.

- **Searching:** This is an operation that searches the specific record using the id or name.
- **Inserting:** The operation is performed when new data needs to be added.
- **Indexing:** It prints all the records in a separate file.
- **Deleting:** This is an operation clears the existing records in various databases

2.2 Requirements

Software Requirement:

Operating system: Windows

2.3Algorithm to build a Primary Index and Secondary Index:

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

Indexes are created using some columns in a file:

- 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.
- The second column is the block where that particular key value can be forced.

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 clustering index. Basically, records with similar characteristics are grouped together and indexes are created for these groups.

The data is sorted according to the search key. It includes sequential file organization. The primary key in data frame is used to create the index. As primary keys are unique and are stored in sorted manner, the performance of searching operation is quite efficient.

Algorithm:

```
Int insert Sorted (int arr [], int n, int key, int capacity)
        If (n >= capacity)
                Return n;
Int I;
        For (1=n-1; (I >=0 && arr[i] >key); i--)
                arr [i+1] = arr[i];
        arr [i+1] = key;

Return (n+1);
```

2.4 Algorithm to Sort the Index based on Primary key:

The Insertion sort, although still o (n2) o (n2), works in a slightly different way. It always maintains a sorted a sub_ list in the lower positions of list. Each new item is then “inserted” Back into the pervious sub -list such that the sorted sub-list is one item larger. We begin by assuming that a list with one item is already sorted. On each pass, one for each item 1 through n-1n-1, the current item is checked against those in the already sorted sub-list. As we look back into the already sorted sub -list, we shift those items that are greater to the right when we reach a smaller item or the end of the sub -list, the current item can be inserted.

The implementation of insertion Sort shows that there we are again n-1n-1 passes to sort n items. The iteration starts at position 1 and moves through position n-1n-1, as these are the items the need to be inserted back into sorted sub-lists. Line 8 performs the shift operation that moves a value up one position in the list, making room behind it for the insertion. Remember that this not a complete exchange as was performed in the previous algorithms.

The, maximum number of comparisons for an insertions sort is the sum of the first n-1n-1 integers. However, in the best case, only one comparison needs to be done on each pass. This would be the case for an already sorted list .One note about shifting versus exchanging is also important. In general, a shift operations requires approximately a third of the processing work of an exchange since only one assignment is performed. In benchmark studies, insert ion sort will allow very good performance.

Algorithm:

```
 if{ high < low }
        return -1;

            int  mid = { low + high} / 2 ;

            if{ key ==  array[mid] }
              return mid ;
               if{key > array[mid]}
                       return binary Search { array,  { mid + 1 } , high , key };
             return binary Search { array , low, { mid -1} , key};
```

2.5 Algorithm for Searching:

When data items are sorted in a collection such as a list, we say that they have linear or sequential relationship. Each data items is stored in a position relative to the others. In java lists, these relative positions are the index values of the individual items. Since these index values are ordered , it is possible for us to visit them in sequence . This process gives rise to our first searching technique, the sequential technique, the SEQUENTIAL SEARCH.

The java implantation for this algorithm is function the needs the list and the items we are looking for and returns a Boolean values as to whether it is present. The Boolean variable found is initialized to False and is assigned the values= True f we discover he items in the list.

Algorithm:

```
SET Lo to 0
SET Hi to array length – 1
WHILE Lo < = Hi
        SET Mid to { Lo + Hi } / 2
        IF X < array [Mid] THEN
             SET Hi to Mid -1
        ELSE IF X > array[Mid] THEN
             SET Lo to  Mid +1
        ELSE
             RETURN Mid
        ENDIF
ENDWHILE
RETURN -1
```

2.6 Calculating time Complexity:

The time complexity of an algorithm is the total amount of time required by an algorithm to complete its execution. In simple words, every piece of code we write, takes time to execute. The time taken by any piece of code to rum is known as the time complexity of that code. The lesser the time complexity, the faster the execution.

The time for program to run does not depend solely on efficiency of code, It's also dependent on the processing power of a PC . Since time complexity is used to measure the time for algorithm, the type of algorithm you'd use in small program wouldn't really matter because there's hardly any work being carried out by the processor although when we write code in professional life, the code isn't of 200 or 300 lines.

It's usually longer than a thesis written by a professor and in cases like that , a lot of processor power is being used.

CHAPTER 3
IMPLEMENTATION

Indexing Orientations

- If n entries have v possible orientations

```
 t=0
for i=1 to n
t = t * v
t = t + or[i]
endfor
return t
```

- To extract the individual orientations again from t < vn-1, use the following code:

```
for i = n to 1
  or[i] = t mod v
  t = t / v
endfor
```

- Usually there is the constraint that the total twist is 0 modulo v, in other words the orientation of the last entry is dependent on the other n-1. In this case just do not encode the orientation of the last piece, which gives a number between 0 and vn-1-1:
- To extract the individual orientations again from t<vn-1-1, use the following code:

```
s = 0
for I = n-1 to 1
        or[i] = t mod v
        s = s – or[i]
        if s < 0 then s = s + v
        t = t / v
endfor
or[n] = s
```

- If n entries can be permuted amongst themselves then their permutation can be encoded in a number between 0 and n!-1. Use a fixed numbering for both the entries and the postitons: the n entries positions are numbered from 1 to n, and the entries are also from 1 to n

```
t = 0;
for I = 1 to n-1
```

```
t = t * (n-i+1)
for j=i+1 to n
if pm[i]>pm[j] then t=t+1
endfor
endfor
retrun t
```

- Note that if all entries are in position then it is encoded as 0. To extract the permutation again from a number t<n! use this:

```
for i=n-1 to 1
pm[i]=1 +(tmod(n-i+1))
t=t/(n-i+1)
for j= i+1 to n
if pm[j]>=pm[i] then pm[j]=pm[i]+1
endfor
endfor
```

- Often there is the constraint that the permutation must have even parity. This means that the position of the last two entries is dependent on the other n-1. In this case just do not encode the position of the last two entries, which gives a number between 0 andn!2-1:

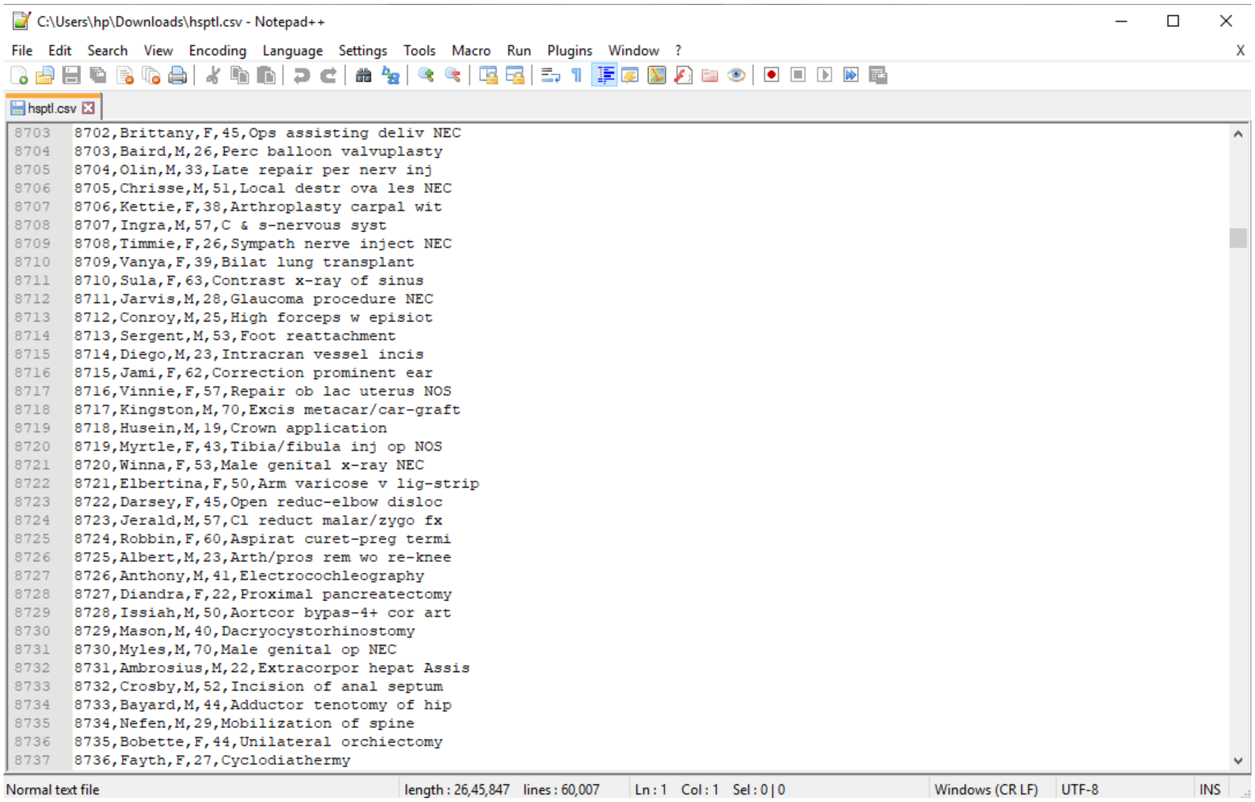
```
t=0;
for i=1 to n-2
t=t*(n-i+)
for j=i+1 to n
if pm[i] > pm[j] then t=t+
endfor
endfor
```

- To extract the even permutation again from a number t>n!/ use this

```
pm[n-1] =1
s=0
for i=n-2 to 1
pm[i]=1 + { t mod (n-i+1)}
s=s+pm[i]-1
```

```
t=t/{n-i+1}
for j=i+1 to n
if pm[j] >=pm[i] then pm[j] = pm[j] +1
endfor
endfor
if s mod 2=1 then swap pm[n],pm[n-1]
Indexing combinations
t=0,r=m
t=0
r=m
for i=n-1 to 0
if cm[i+1] =p then
t=t+ comb(i,r)
r=r-1
endif
next
return t
r=m
```

CHAPTER 4
RESULTS AND SNAPSHOTS




```
8702,Brittany,F,45,Ops assisting deliv NEC
8703,Baird,M,26,Perc balloon valvuplasty
8704,Olin,M,33,Late repair per nerv inj
8705,Chrisse,M,51,Local destr ova les NEC
8706,Kettie,F,38,Arthroplasty carpal wit
8707,Ingra,M,57,C & s-nervous syst
8708,Timmie,F,26,Sympath nerve inject NEC
8709,Vanya,F,39,Bilat lung transplant
8710,Sula,F,63,Contrast x-ray of sinus
8711,Jarvis,M,28,Glaucoma procedure NEC
8712,Conroy,M,25,High forceps w episiot
8713,Sergent,M,53,Foot reattachment
8714,Diego,M,23,Intracran vessel incis
8715,Jami,F,62,Correction prominent ear
8716,Vinnie,F,57,Repair ob lac uterus NOS
8717,Kingston,M,70,Excis metacar/car-graft
8718,Husein,M,19,Crown application
8719,Myrtle,F,43,Tibia/fibula inj op NOS
8720,Winna,F,53,Male genital x-ray NEC
8721,Elbertina,F,50,Arm varicose v lig-strip
8722,Darsey,F,45,Open reduc-elbow disloc
8723,Jerald,M,57,Cl reduct malar/zygo fx
8724,Robbin,F,60,Aspirat curet-preg termi
8725,Albert,M,23,Arth/pros rem wo re-knee
8726,Anthony,M,41,Electrocochleography
8727,Diandra,F,22,Proximal pancreatectomy
8728,Issiah,M,50,Aortcor bypas-4+ cor art
8729,Mason,M,40,Dacryocystorhinostomy
8730,Myles,M,70,Male genital op NEC
8731,Ambrosius,M,22,Extracorpor hepat Assis
8732,Crosby,M,52,Incision of anal septum
8733,Bayard,M,44,Adductor tenotomy of hip
8734,Nefen,M,29,Mobilization of spine
8735,Bobette,F,44,Unilateral orchiectomy
8736,Fayth,F,27,Cyclodiathermy
```

Fig 4.1 The figure contains hospital dataset which consists of Id, Name, Gender, Age and description

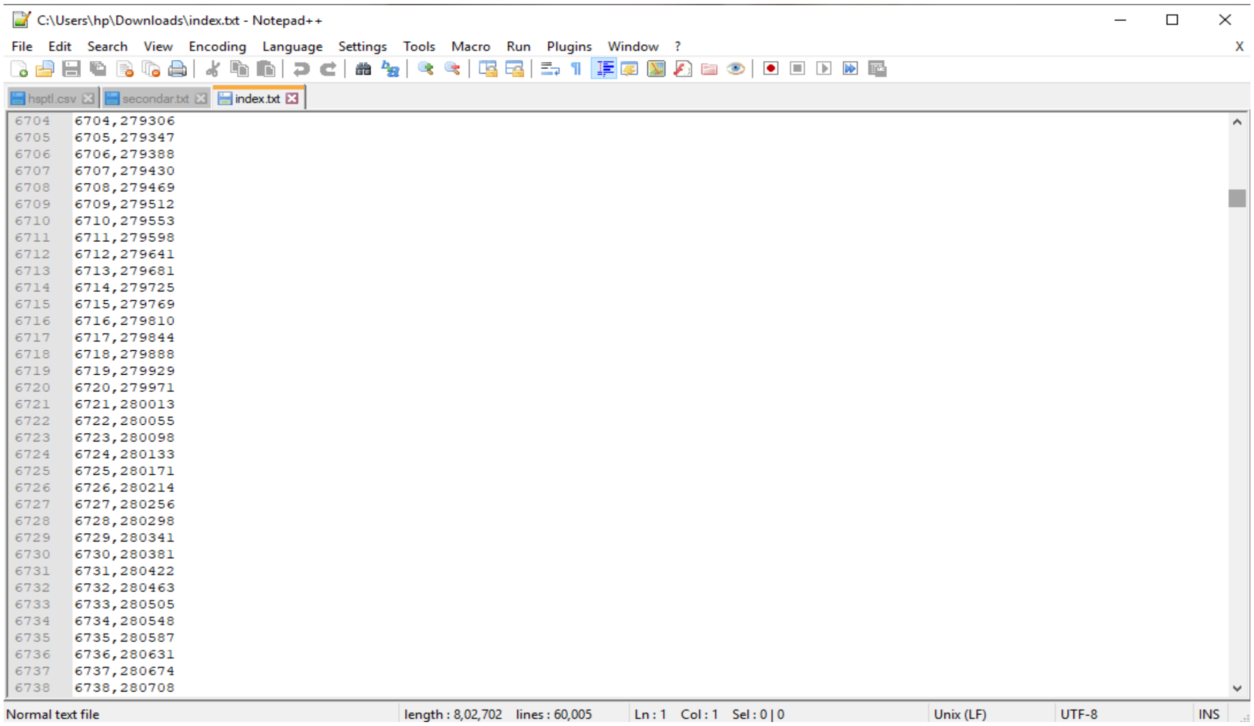


Fig 4.2 Index build using primary key, here the primary key is Id

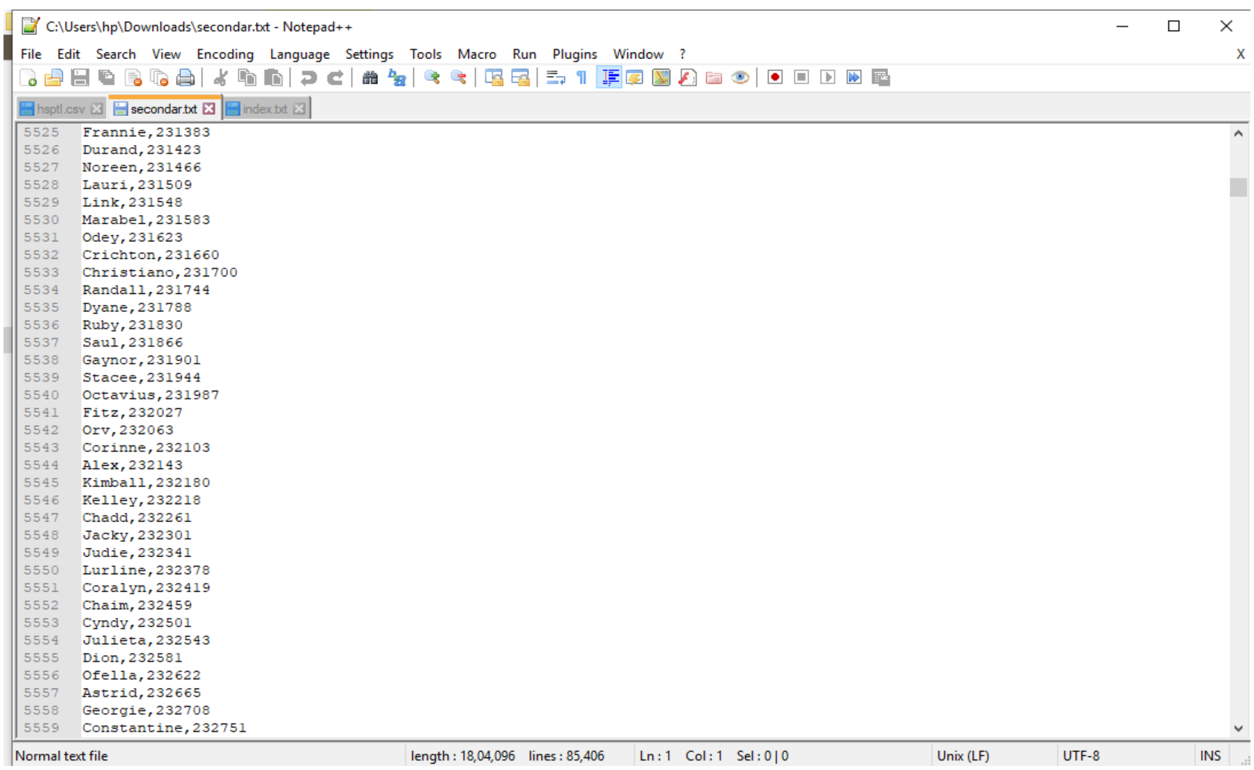


Fig 4.3 Index build using secondary index, here secondary key is Name

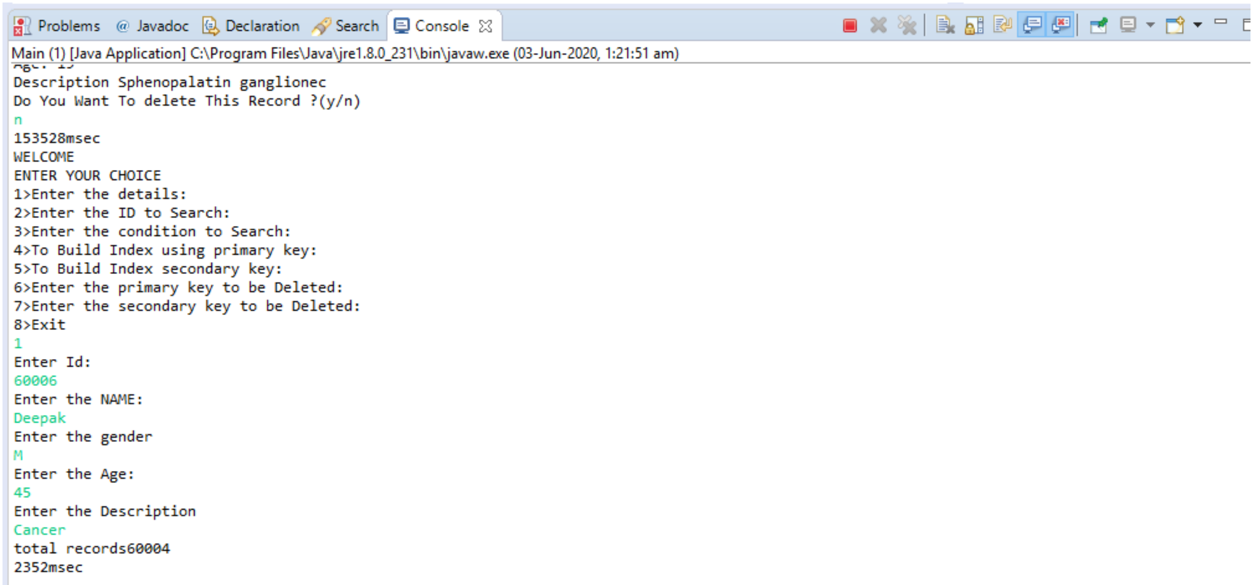


Fig 4.4 Insertion of the record into the data
Inserts the new records to the dataset

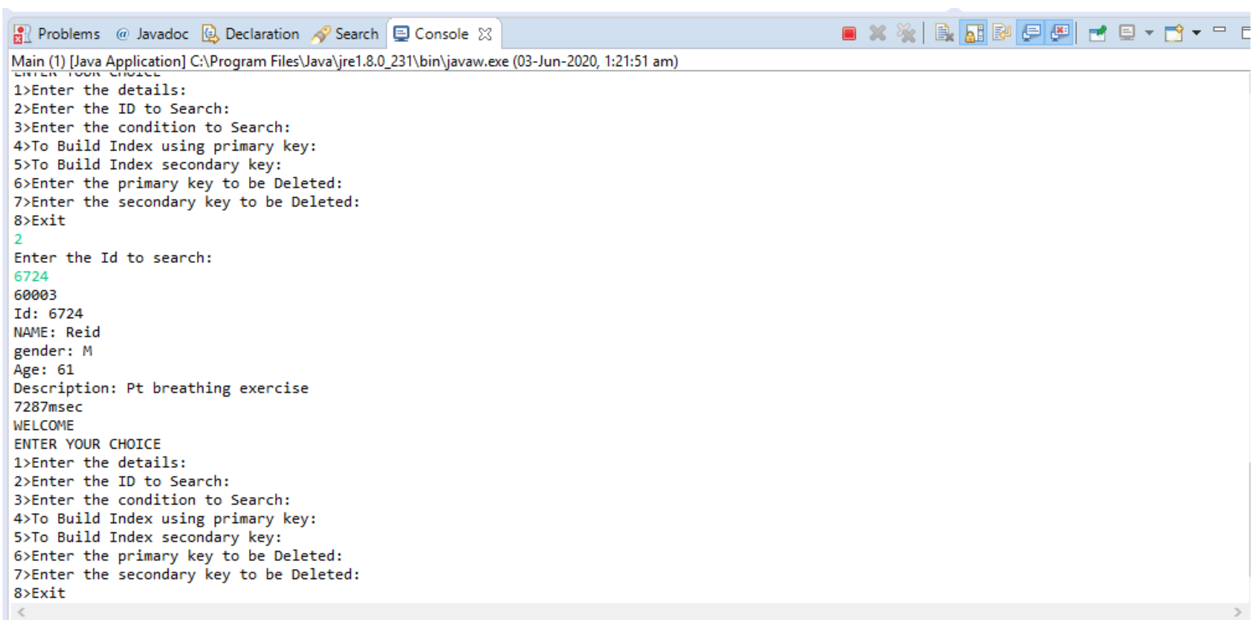


Fig 4.5 searching of the primary key
Searches for the primary key that is id in the dataset

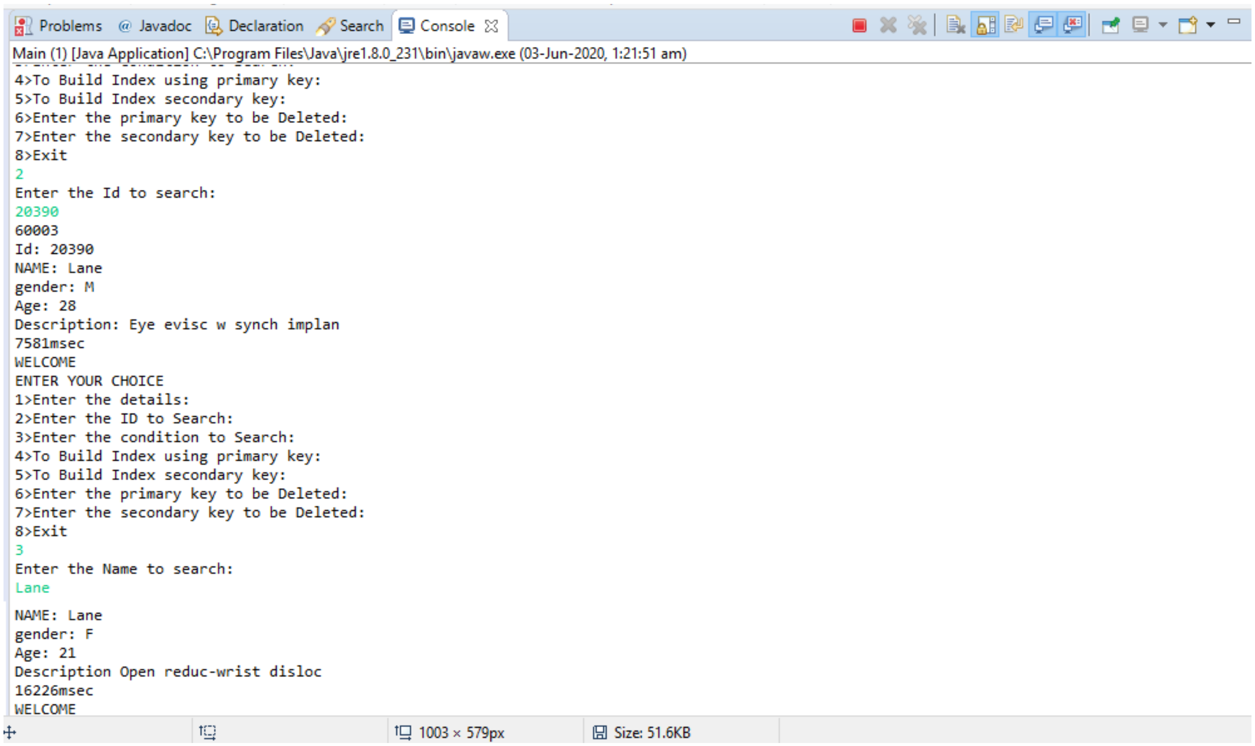


Fig 4.5 Searching of secondary key
Searches for the secondary key that is name in the dataset

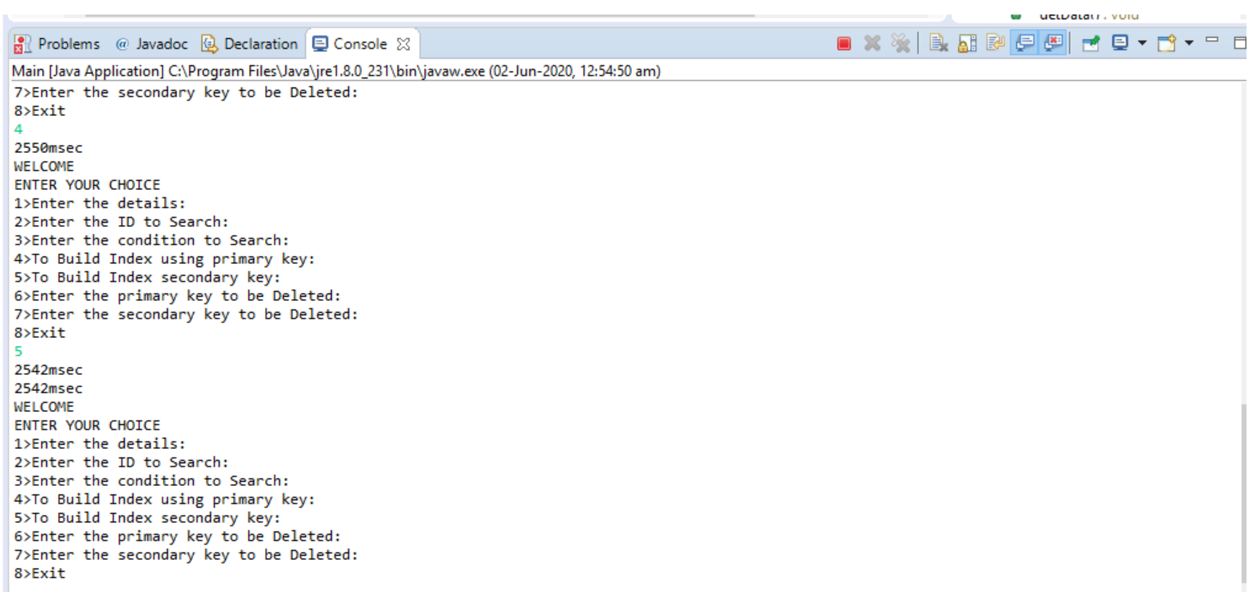


Fig 4.6 Building the Index for the data

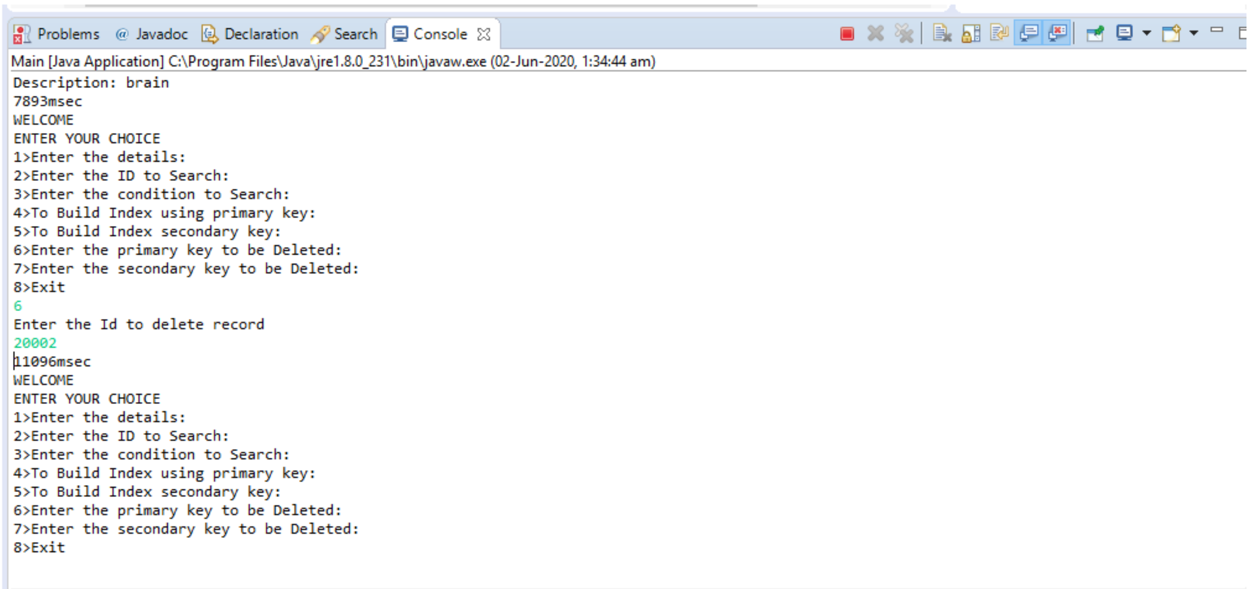


Fig 4.7 Deletion of a Primary key

Deletes the primary key which is id in the data

Analysis

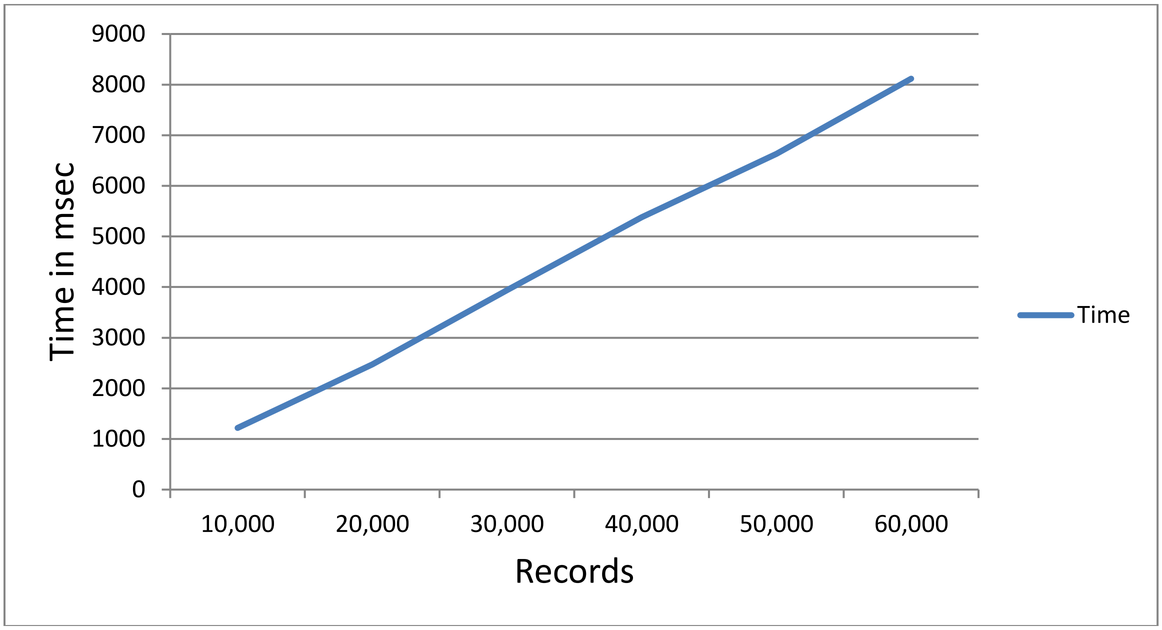


Fig 4.8 Time taken to read the data

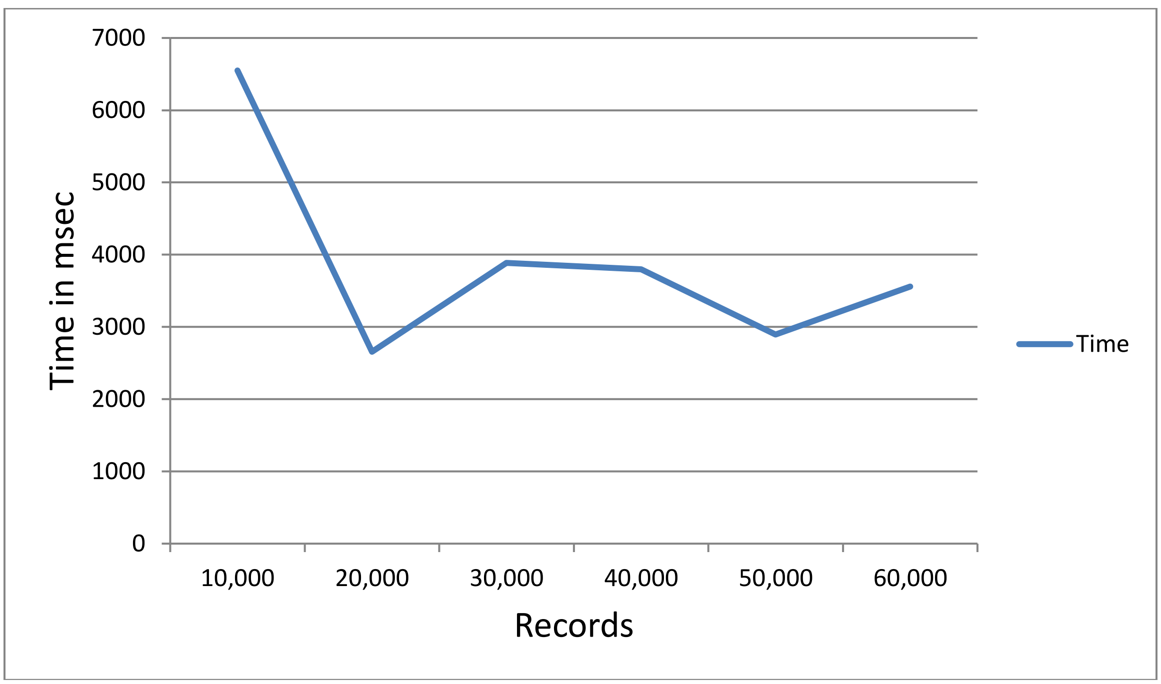


Fig 4.9 Time taken to search primary key in the data

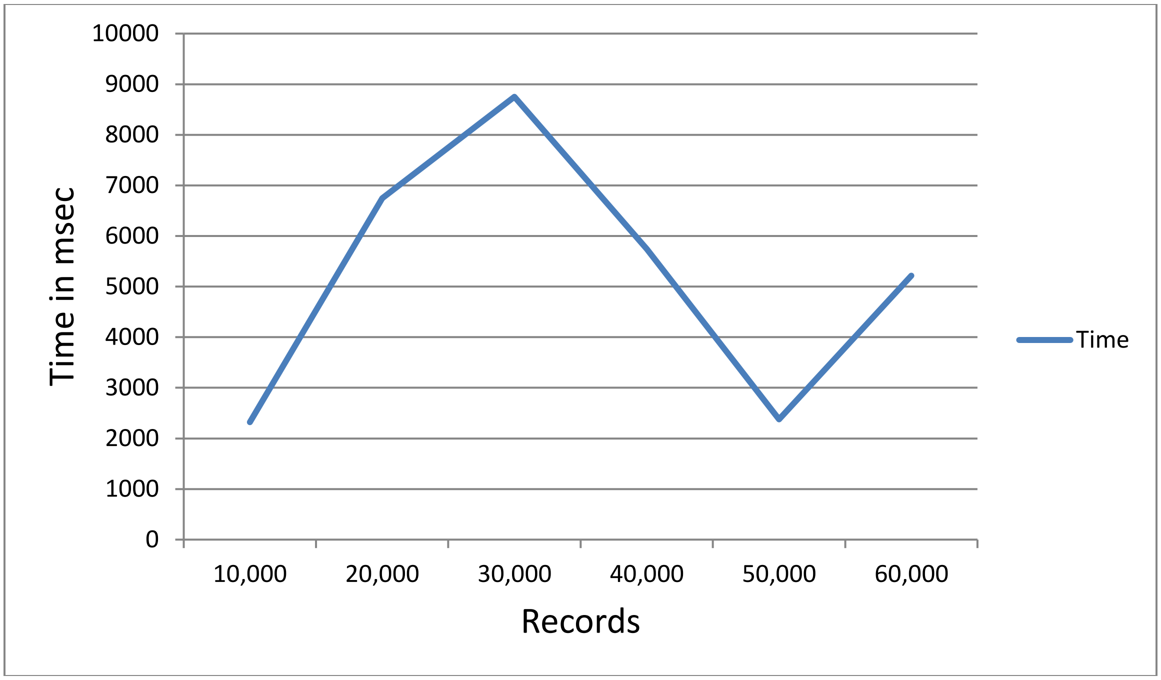


Fig 4.10 Time taken to search secondary index

The above figure indicates the time analysis the searching of the record in the secondary index

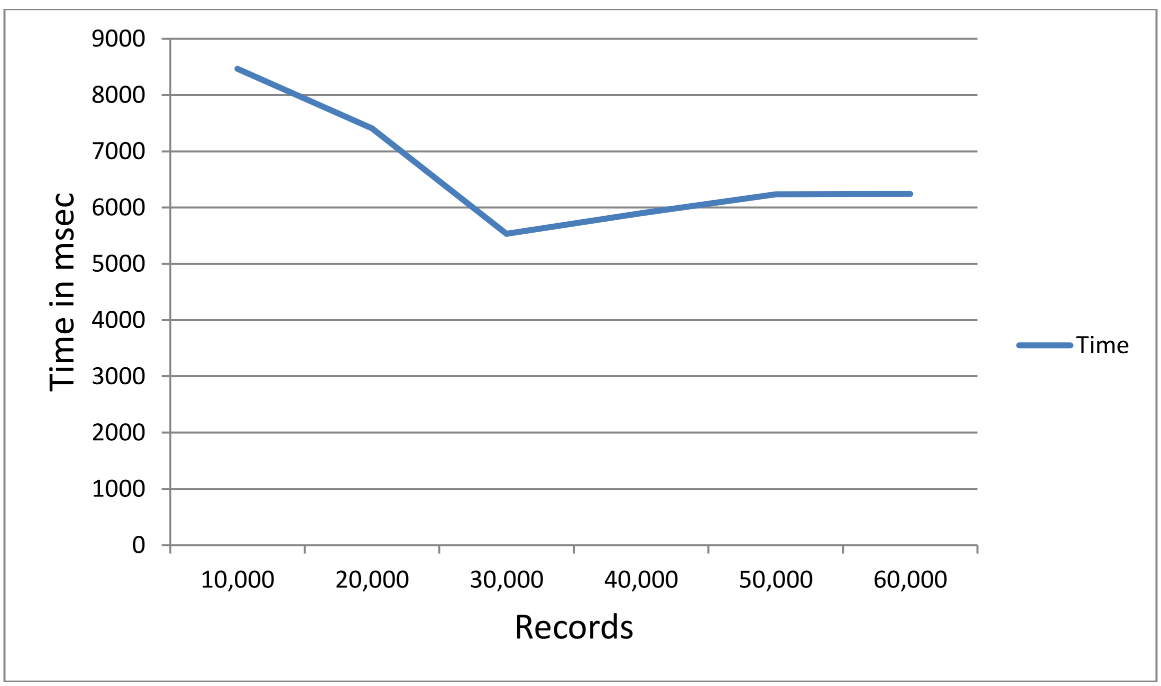


Fig 4.11 Time taken for deleting primary index

The above figure indicates the time analysis for deleting the record from the dataset

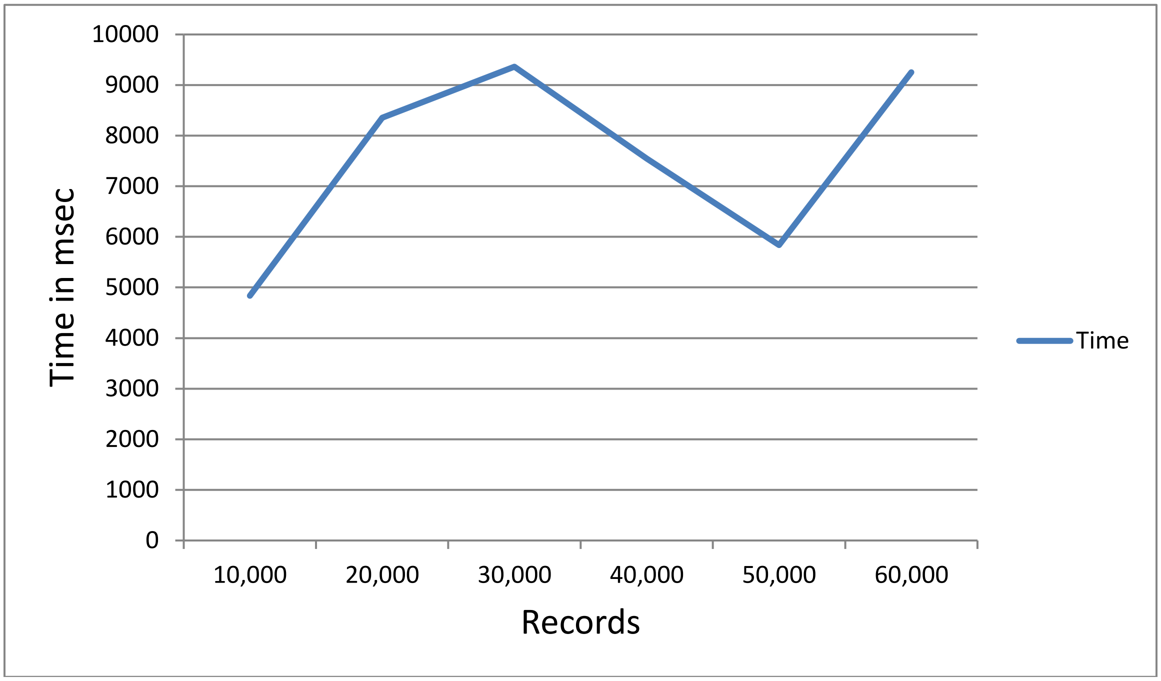


Fig 4.12 Time taken for deleting the secondary key

The above figure indicates the time analysis deleting the secondary key from the dataset

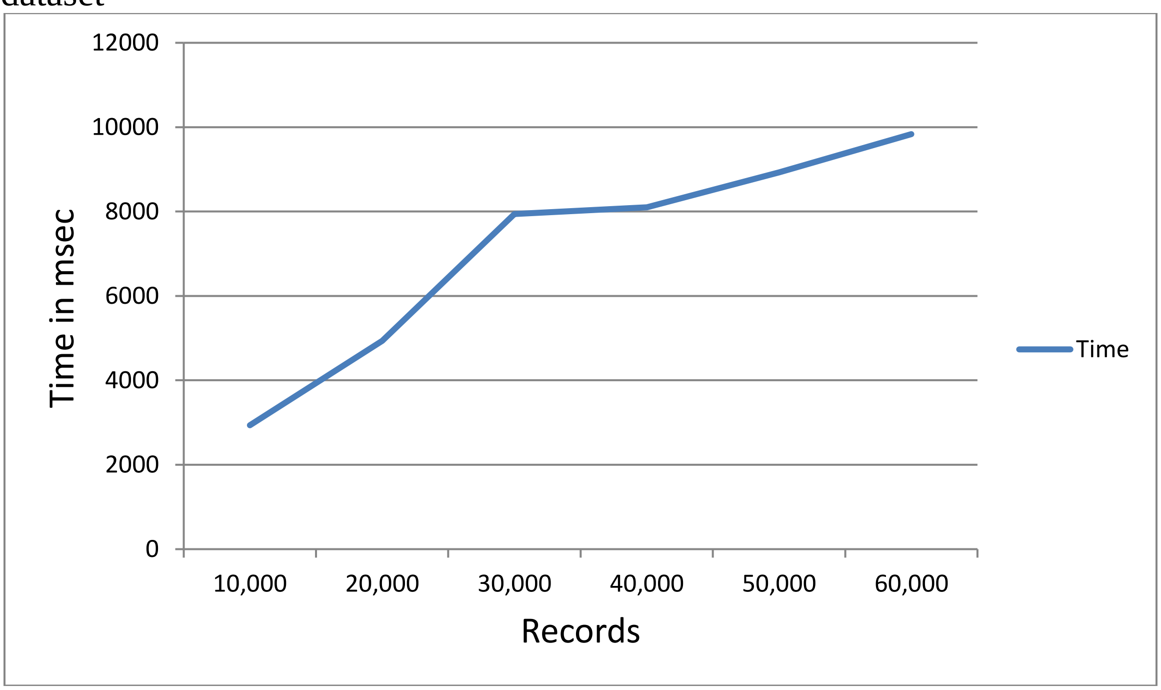


Fig 4.13 Time taken to insert the records.

The above figure indicates the time analysis for inserting the record into the file

CONCLUSION

This project is only a humble venture to satisfy the needs to manage the project work. Effective implementation of this project will take care of the basic requirements of the hospital management system because it is capable of providing easy and effective storage of information related to activities happening in the stipulated area. With these, the objectives of the system design will be achieved.

In order to allow for future expansion, the system has been designed in such a way that will allow possible modification as it may deem necessary by the hospital management, whenever the idea arises.

We have successfully used various functionalities of JAVA and created the file structures.
Indexes form an important part of designing, creating and testing information.
Users search in a hurry for information to help them and give up after two or three tries.
An index can point the way in harmony with user expectations or not.
Indexing is an interactive analysis and creative process throughput the entire documentation.

Features:

1. Clean separation of various components to facilitate easy modification and revision.
2. All the data is maintained in a separate file to facilitate easy modification.
3. All the data required for different operations is kept in a separate file.

REFERENCES

- www.kaggle.com
- www.javatpoint.com
- w3chools.com
- www.geeksforgeeks.com
- www.wikipedia.com