## Visvesvaraya Technological University Belagavi-590 018, Karnataka



A Mini Project Report on

## “Indexing for gun violence dataset”

## Mini Project Report submitted in partial fulfilment of the requirement for the File Structures Lab [15ISL68]

**Bachelor of Engineering In**

**Information Science and Engineering Submitted by**

**Ananya SG [1JT17IS005] Under the Guidance of Mr.Vadiraja A**

**Asst. Professor Dept. Of ISE**



**Department of Information Science and Engineering Jyothy Institute of Technology**

**Tataguni, Bengaluru-560082 2019-20**

**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 Ananya SG [1JT17IS005]** bonfire 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 **2018-2019**. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the Report deposited in the departmental library. The project report has been approved as it satisfies the academic requirements in respect of Project work prescribed for the said Degree.

### Mr Vadiraja A Dr. Harshvardhan Tiwari

Guide, Asst. Professor Associate. Professor and HOD

Dept. Of ISE Dept. Of ISE

External Viva Examiner Signature with Date: 1.

2.

## ACKNOWLEDGEMENT

Firstly, we are very grateful to this esteemed institution **“Jyothy Institute of Technology**” for providing us an opportunity to complete our project.

We express our sincere thanks to our Principal **Dr. Gopalakrishna K for** providing us with adequate facilities to undertake this project.

We would like to thank **Dr. Harshvardhan Tiwari , Associate Prof. and Head** of Information Science and Engineering Department for providing for his valuable support.

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

Finally, we would thank all our friends who have helped us directly or indirectly in this project.

Ananya SG[1JT17IS005]

**ABSTRACT**

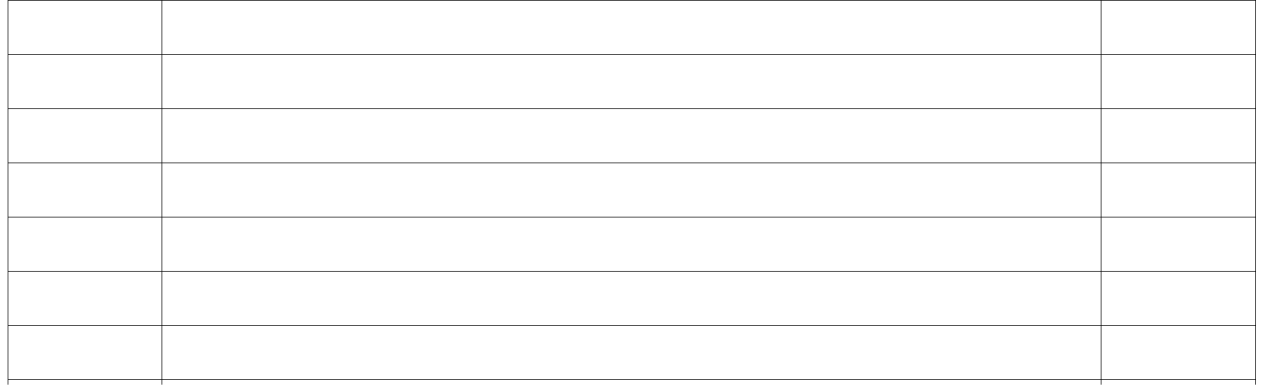
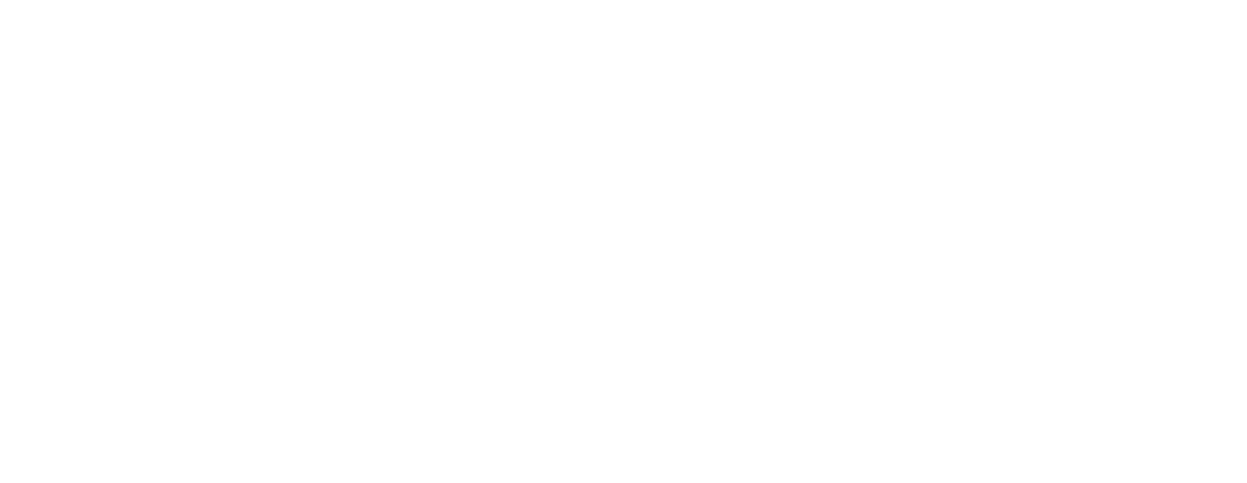
In this mini project I have created one application which is easy to access and user friendly. For this application is developed using simple java code. The project title is “Indexing” which is used in the application and for distinguishing the type of file structure to be used.

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.

Indexing is a data structure technique to efficiently retrieve records from the files based on some attributes on which the indexing has been done. An index in which the entries are a key ordered liner list. Simple indexing can be useful when the entire index can be held in memory. Changes like addition and searching require both the index and data file to be changed.

Updates affect the index affect the index if the key field is changed, or if the record is moved. An update which moves a record order is determined by the order in which they are entered. The physical order of records in the file may not be the dame as order of entry, because of record deletions and space reuse. The index should be read into memory when the data file is opened.

**Table of Contents**



|  |  |  |
| --- | --- | --- |
| **SL No** | **Description** | **Page No.** |
| 1. | Introduction | 1-3 |
| 2. | Algorithm | 4-7 |
| 3. | Implementation | 8-10 |
| 4. | Results and Screenshots | 11-22 |
| 5. | Conclusion | 25 |

6. Reference 26

CHAPTER 1

INTRODUCTION

**Introduction to File Structure**

A database is simply on organized collection of related data, typically stored on disk, and accessible by possibly many concurrent users. Databases are generally separated into application areas. For example, one database may contain Human Resource (employee and payroll) data; another may contain sales data; another may contain accounting data; and so on. Databases are managed by a DBMS.

In computing, a file system or filesystem controls how data is stored and retrieved. Without a file system, information placed in a storage medium would be one large body of data with no way to tell where one piece of information stops and the next begins. By separating the data into pieces and giving each piece a name, the information is easily isolated and identified. Taking its name from the way paper-based information systems are named, each groups of data is called a “file”. The structure and logic rules used to manage the groups of information and their names is called a “file system”.

There are many different kinds of file systems. Each one has different structure and logic, properties of speed, flexibility, security, size and more. Some file systems have been designed to be used for specific applications. For example, the ISO 9660 file system is designed specifically for optical discs.

File systems can be used on numerous different types of storage devices that use different kinds of media. The most common storage device in use today is a hard disk drive. Other kinds of media that are used include flash memory, magnetic tapes, and optical discs. In some cases, such as the computers main memory (random-access memory, RAM) is used to create a temporary file system for short-term use.

Some file systems are used on local data storage devices; others provide file access via a network protocol(for example, NFS, SMB, or 9P clients). Some file systems are “virtual”. Meaning that the supplied “files” (called virtual files) are computed on request (e.g. procfs) or are merely a mapping into a different file system used as a blacking store. The file system manages access to both the content of files and the metadata about those files. It is responsible for arranging storage space; reliability, efficiency, and tuning with regard to the physical storage medium are important design considerations.

1

**Introduction to JAVA**

1. 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.
2. 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.
3. Structurally, Java language starts with packages. A packages is the Java languages 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.

**Indexing**

Index: 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.

1. An index imposes order on a file without rearranging the file.
2. Indexing works by indirection.
   1. A Simple Index for Entry-Sequenced Files
   2. An index in which the entries are a key ordered linear list.
3. Simple indexing can be useful when the entire index can be held in memory.
4. Changes (additions and deletions) require both the index and the data file to be changed.
5. Updates affect the index if the key field is changed, or if the record is moved.
6. An update which moves a record can be handled as a deletion followed by an addition
7. Object Oriented Support for Indexed, Entry Sequenced Files
8. A file in which the record order is determined by the order in which they are entered
9. The physical order of records in the file may not be the same as order of entry, because
10. of record deletions and space deletions.
11. The index should be read into memory when the data file is opened.
12. Indexes That are too Large to Hold in Memory

2

1. Searching of a simple index on disk takes too much time.
2. Maintaining a simple index on disk in sorted order takes too much time.
3. Tree structured indexes such as B-tress are a scalable alternative to simple indexes.
4. Hashed organization is an alternative to indexing when only a primary index is needed.
5. Indexing to Provide Access by Multiple Keys
6. A search key other than the primary key.
7. An index built on a secondary key.
8. Secondary indexes can be built on any field of the data file, or on combinations of fields.
9. Secondary indexes will typically have multiple locations for a single key.
10. Changes to the data may now affect multiple indexes.
11. The reference field of a secondary index can be a direct reference to the location of the entry in the data file.
12. The references fields of a secondary index can also be an indirect reference to the location of the entry in the data file, through the primary key.
13. Indirect secondary key references simplify updating of the file set.
14. Indirect secondary key references increases access time.
15. Retrieval Using Combinations of Secondary Keys
16. The search for records by multiple keys can be done on multiple index, with the combination of index entries defining the records matching the key combination.
17. If two keys are to be combined, a list of entries from each key index is retrieved.
18. For an “or” combination of keys, the lists are merged.
19. I.e., only entries found in entire list matches the search.
20. For an “and combination of keys, the lists are matched.
21. I.e., only entries found in both the lists match the search.

3

**CHAPTER 2**

**ALGORITHM**

**Algorithm 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 array [], int n, int key, int capacity)

if (n >= capacity)

Return n;

int i;

For (i=n-1; (i >=0 && array[i] >key); i--)

array [i+1] = array[i];

array [i+1] = key;

Return (n+1);

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 (position 00) 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. Again this is o(n2)o(n2). However, in the best case, only one comparsion 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(low>high)

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);

**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.

5

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 Low to 0

SET High to array length – 1

WHILE Lo < = Hi

SET Mid to (Low + High) / 2

IF X < array [Mid] THEN

SET Hi to Mid -1

ELSE IF X > array [Mid] THEN

SET Low to Mid +1

ELSE

RETURN Mid

ENDIF

ENDWHILE

RETURN -1

**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.

If you’ve programmed a bit before, you’re probably wondering how this can be of any sue for you because your program was running fin even when you didn’t know all of this time complexity stuff, right? I agree with you 100% but there’s a catch.

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 is not of 200 to 300 lines.

6

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. if your code is efficient in terms of data structures , you might find yourself in a rather sticky situation.

7

**CHAPTER 3**

**IMPLEMENTATION**

**IMPLMENTATION**

Indexing Orientations

* If n entries have v possible orientations

t=0

for i=1 to n

t = t \* v

t = t + or[i]

end for

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

end for

* 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

end for

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 positions : 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)

8

for j=i+1 to n

if pm[i]>pm[j] then t=t+1

end for

end for

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

end for

end for

* 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+

end for

end for

* 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

9

if pm[j] >=pm[i] then pm[j] = pm[j] +1

end for

end for

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

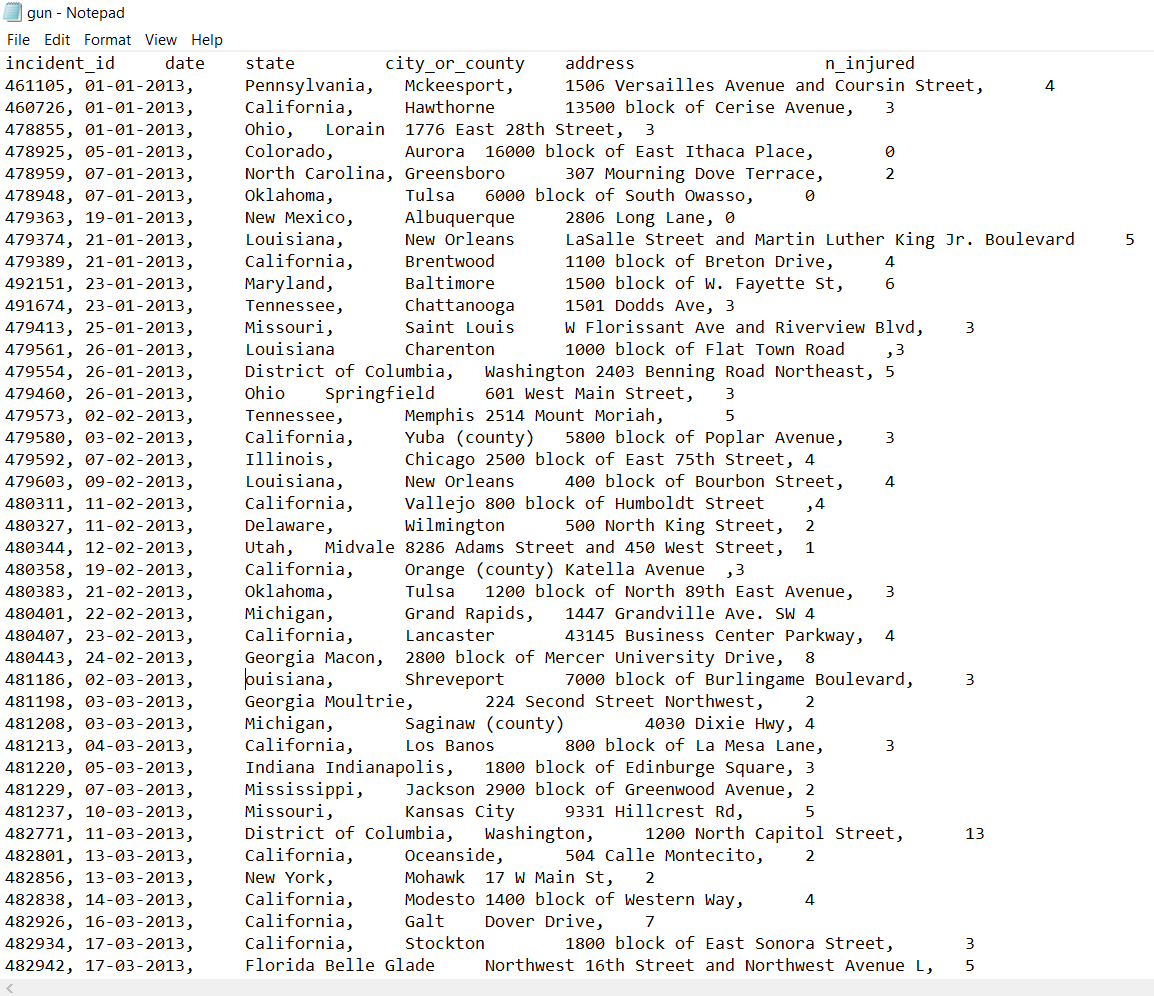
return t

r=m

10

**CHAPTER 4**

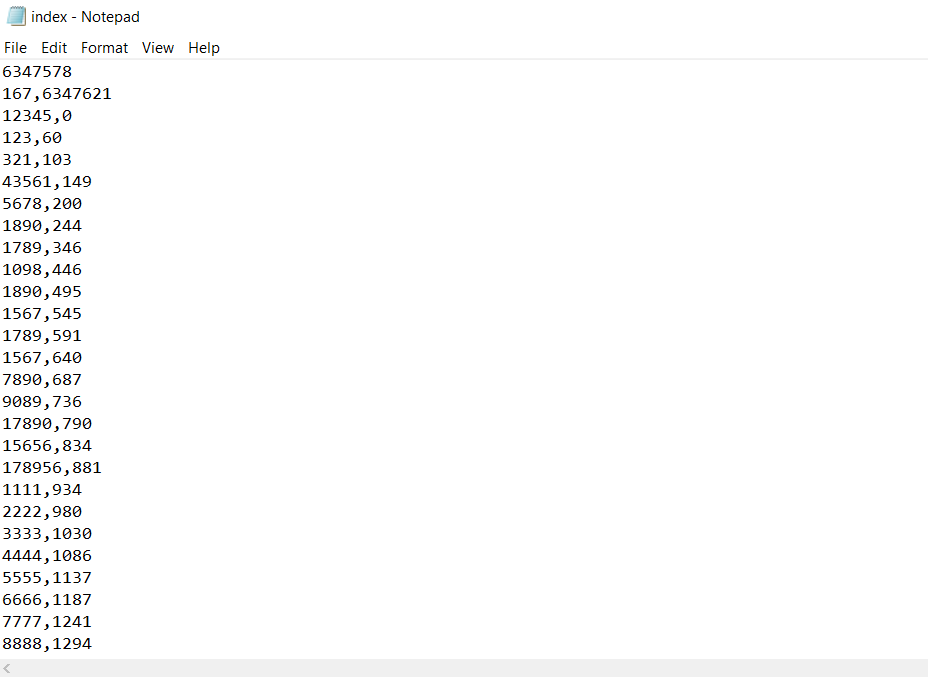
**RESULTS AND SCREENSHOTS**



**Fig4.1: Gun Violence Dataset**

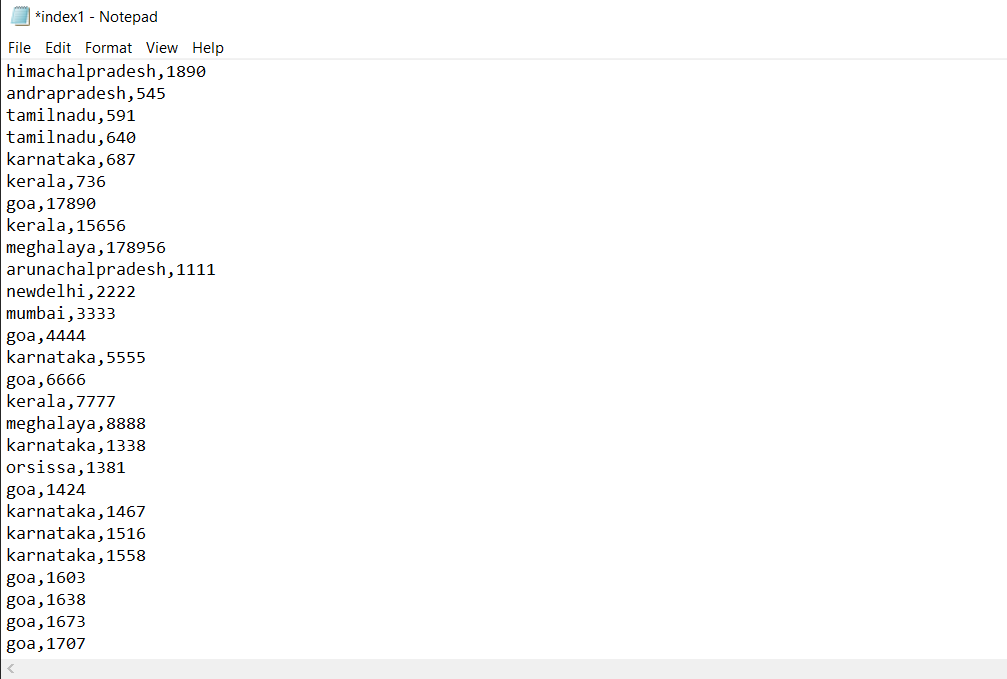
The above dataset consists of incident id, date, state, city or address, n injured. This is the dataset which is further get indexed.

11



**Fig 4.2** index build for gun violence dataset using primary key incident id

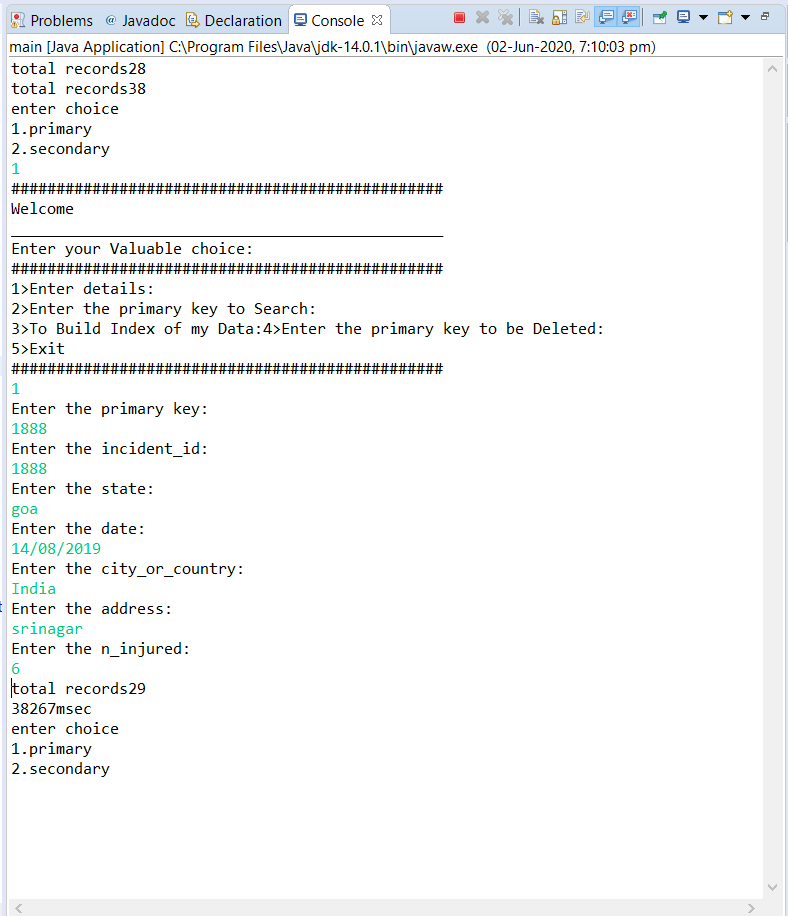
The index is built for the primary key id in the dataset.



**Fig 4.3** index build for toy dataset using secondary key(country)

The index is built for the secondary key country in the dataset

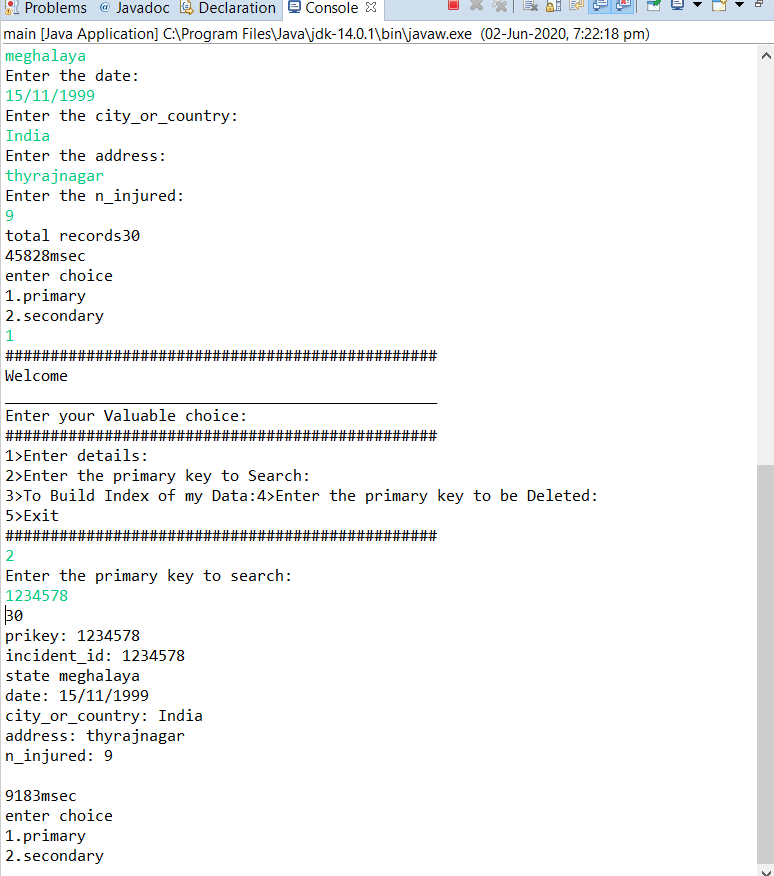
12



**Fig4.3** Insertion of data

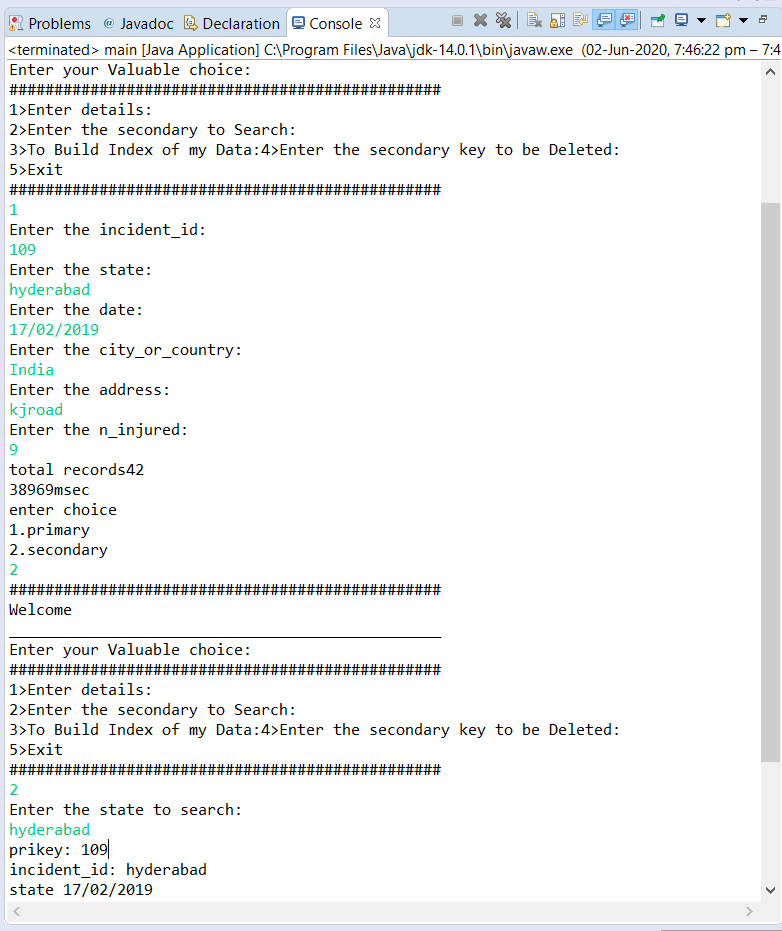
Details are entered here and appended to the dataset

13



**Fig 4.4** Searching of primary key in data

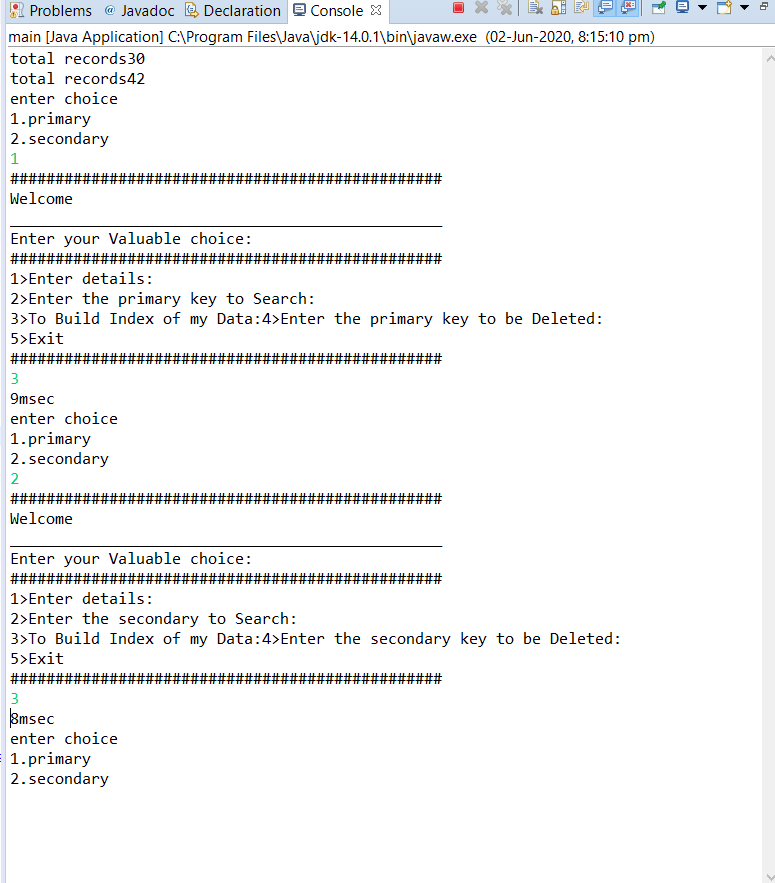
14



**Fig 4.5** Searching a record using secondary key(state)

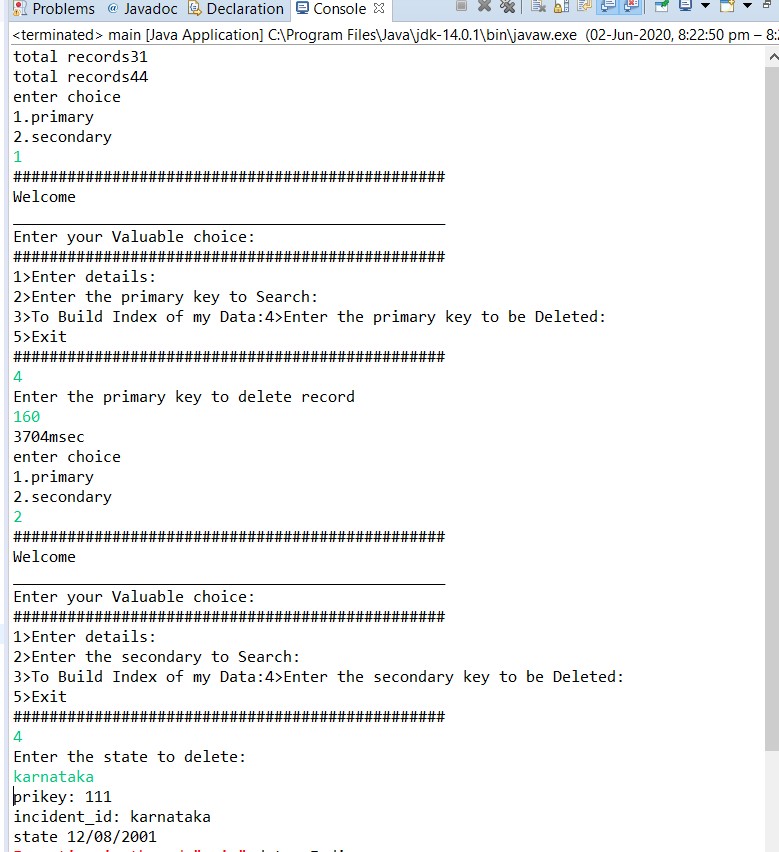
To search the details of toy export, secondary key i.e., country has been entered to extract the details.

16



**Fig 4.6** Building a index of a data

17



**Fig 4.7** Deleting primary key(incident id) and secondary key(country) record in a file

18

**Time Analysis For Inserting the Record**

The time taken to insert a record into a data file, as the number of record increases the time also increases.

**Time analysis for searching a primary key record**

The time taken to search a primary key record in the data file, as the number of record increases, the time will be delayed to search a record.

19

**Time Analysis for Deleting a primary key record**

The time taken to delete from the data file, decreases as the number of record increases.

**Time Analysis for searching a secondary key record**

The time taken to search the record in a data file, as the number of record increases the time will delayed to search of record.

**Time Analysis for inserting a secondary key record**

20

The time taken to insert a record into a data file, as the number of record increases the time also increases.

21

**Time Analysis for Deleting a secondary key record**

The time taken to delete from the data file, decreases as the number of record increases.

22

**CONCLUSION**

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.

View tables are used to display all the components at once so that user can see all the components of a particular type at once. One can just select the component and modify and remove the component.

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.

23

**REFERENCES**

* [www.kaggle.com](http://www.kaggle.com)
* [www.geeksforgeeks.com](http://www.geeksforgeeks.com)
* [www.javaTpoint.com](http://www.javaTpoint.com)
* [www.github.com](http://www.github.com)

24