**B-Tree Project (Index Seek Implementation)**

**Summer Internship Report**

Submitted in partial fulfilment of the requirement for the degree of

Bachelor of Technology

in

Information Technology

by

**Guneet Malhotra – 01496303119**

****

Maharaja Surajmal Institute of Technology

(Affiliated to Guru Gobind Singh Indraprastha University)

Janakpuri, New Delhi-58

October-2021

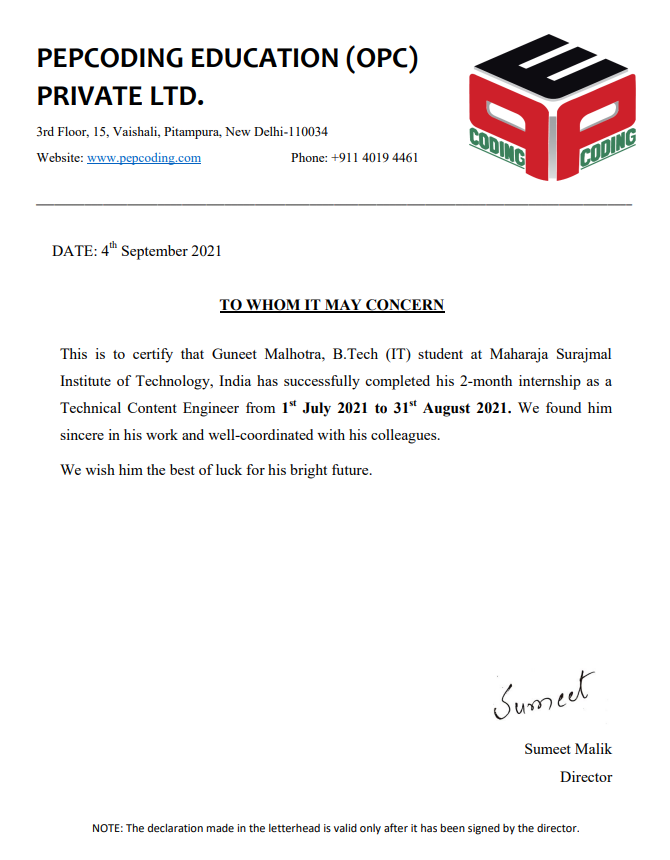
**Student Certificate**

I, GUNEET MALHOTRA, Enrolment No. – 01496303119, student of Maharaja Surajmal Institute of Technology, New Delhi, BTech (5th semester) hereby declare that the project report is based on my own work carried out during the course of internship. This report has not been submitted to any other institute for the award of any other degree/diploma/certificate. I further certify that I have followed the guidelines provided by the college in writing the report.

Guneet Malhotra

Enrolment Number- 01496303119

IT-Evening 5th Semester

****

**Acknowledgement**

First of all, I would like to thank Mr Sumeet Malik, Director of Pepcoding Education (OPC) Private Ltd, to give me the opportunity to work within this precious organisation.

I would also like to thank Mr Madhusudan Malik, my mentor during the whole internship process, who guided me on every big and small step and made this journey fruitful and smooth for me.

I would also like to thank my college, Maharaja Surajmal Institute of Technology’s management and staff for providing me the right guidance and opportunities and especially the teachers and Head of Department of Information Technology branch, for showing me the right path.

I would also like to thank my peers at Pepcoding Education for constantly motivating me and supporting me throughout this journey.

Last but not the least, I would like to thanks my friends and family who have always believed in me and have provided me constant moral support and guidance.

Guneet Malhotra

01496303119

**Table of contents**

Title Page I

Student Certificate II

Organisation Certificate III

Acknowledgement IV

Abstract VIII

Chapter-1 Company Profile 1

Chapter-2 Technologies and Tools 3

2.1 Java

2.1.1 Advantages of Java 3

2.1.2 Applications of Java Programming 7

2.2 Java Swing

2.2.1 Difference b/w AWT and Swing 8

2.2.2 MVC Architecture 9

2.2.3 Swing Features 10

Chapter-3 Demonstration of Technology through Project

3.1 Problem Description 11

3.2 Objective 11

\*\* DBMS FILE ORGANISATION \*\*

3.3 Heap File Organisation 12

3.4 Sequential File Organisation 12

3.5 Hash File Organisation 13

3.6 Clustered File Organisation 13

3.7 File Operations 13

3.7.1 Update Operation 13

3.7.2 Retrieval Operation 13

3.8 DBMS Data Storage System

3.8.1 Primary Storage 14

3.8.2 Secondary Storage 15

3.8.3 Tertiary Storage 15

\*\* STUDY OF INDEXING \*\*

3.9 Indexing

3.9.1 Dense Index 16

3.9.2 Sparse Index 17

3.9.3 Multilevel Index 17

\*\* B TREE \*\*

3.10 Introduction 19

3.11 Operations 20

3.11.1 Searching 20

3.11.2 Inserting 20

3.11.3 Deletion 21

3.12 Applications of B-Tree 23

\*\*IMPLEMENTATION DETAILS \*\*

3.14 Implementation Blocks 24

3.14.1 Insertion 25

3.14.2 Searching 25

3.14.3 Code Structure 25

Chapter-4 Screenshots and Working of the Project

4.1 Running The Project

4.1.1 Command Line Arguments 26

4.1.2 App Launches 27

4.2 Home Panel 29

4.3 Data Panel 31

4.4 Index Panel 36

4.5 Query Panel 43

Chapter-5 Conclusion 48

Bibliography 49

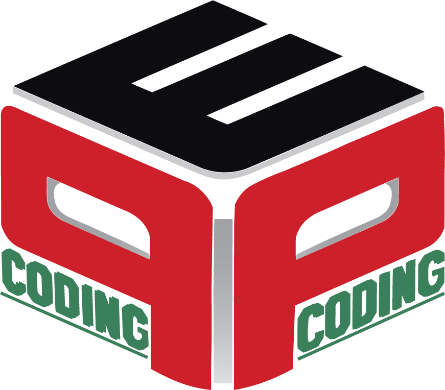
**Abstract**

This is a study-based project and the basic point motivating this project is learning and implementing algorithms that reduces time and space complexity.

In this project, we reduce the time taken to search a given record by using a B/B+ tree rather than indexing and traditional sequential access. It is concluded that disk-access times are much slower than main memory access times. Typical seek times and rotational delays are of the order of 5 to 6 milliseconds and typical data transfer rates are of the range of 5 to 10 million bytes per second and therefore, main memory access times are likely to be at least 4 or 5 orders of magnitude faster than disk access on any given system. Therefore, the objective is to minimize the number of disk accesses and thus, this project is concerned with techniques for achieving that objective i.e. techniques for arranging the data on a disk so that any required piece of data, say some specific record, can be located in a few I/O’s as possible.

**Chapter 1**

**Company Profile**



**(Fig 1.1: Pepcoding Official Logo)**

**Company Name:** Pepcoding Education (OPC) Private Ltd.

**Website Link:** [**https://www.pepcoding.com**](https://www.pepcoding.com)

**YouTube Link:** [**Pepcoding YouTube Channel**](https://www.youtube.com/c/Pepcoding)

**LinkedIn Link:** [**Pepcoding LinkedIn**](https://www.linkedin.com/company/pepcoding-education/)

**Current Location (Main Office):** Noida Sector-63, Uttar Pradesh, India

**History:** Pepcoding was started by Mr Sumeet Malik in December 2017. It is an educational institute which was started with the aim of Pursuit of Peace and Excellence (PEP) in Coding thus named PEPCODING. The first office of Pepcoding was set up in Pitampura, Delhi.

**Aims and Objectives:** The aim of Pepcoding is to bring the coding culture to every college and every student in those colleges and that too free of cost. Pepcoding believes that everything can be achieved with mere hard work and regardless of today’s thinking, we should do not just smart work but a lot of hard work too.

**Work and Achievements:**  Over the time span of past 2-3 years, Pepcoding has been able to place a lot of students into tech giants like Amazon, Microsoft, Google, LinkedIn, HashedIn, Goldman Sachs, Adobe, Intuit, Cuemath, Walmart, IBM Labs, Capgemini etc.

Not only this, since the lockdown period and the global pandemic COVID-19 hit the world, Pepcoding has given all it’s content free of cost to the entire coding community through their website and YouTube channel. Pepcoding has 3 courses named Data Structures and Algorithms Levels 1, 2 and 3 which have in total more than 1200 questions with programming solutions available in C++ and Java.

There are video solutions to all the 1200 problems in Java. Not only this, Pepcoding has made all the web development courses free of cost too.

As per scale growth of Pepcoding as a company, they have build a very large community of 20 thousand on LinkedIn and still counting. Pepcoding has a coding family of 81 thousand and still counting on YouTube.

Recently Pepcoding has achieved a total of approximately **121% headcount growth** in just 6 months. Pepcoding has also launched a new idea in the form of a new website named [**NADOS**](https://nados.pepcoding.com) (Not a degree-only skills) empowering the importance of skills as only a degree cannot fulfil the required employment criteria nowadays.

Also, Pepcoding has recently launched the [**app NADOS**](https://play.google.com/store/apps/details?id=com.nados) as a social media platform for the coding community where the coders can post their project ideas, make videos on coding and teach and help each other, create rooms and conduct virtual mock interviews with each other.

Pepcoding also has a tie-up with more than 20 tech companies for placements and have and will conduct various placement drives for students free of cost i.e. even the students who are not a paid member of Pepcoding can grab the opportunity of a good placement just on the basis of their skills.

**Future aims:** In the future, Pepcoding aims at expanding in every domain of Education and provide everything free of cost to the students so that no student will lack an opportunity to achieve something big in his/her life. Pepcoding aims at providing free of cost education in coding, Science, arts, law etc. and aims at helping the students in every possible way.

**Chapter-2**

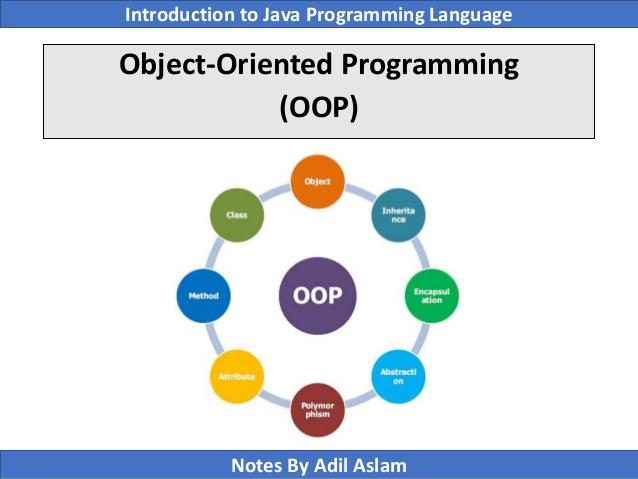
**Technology and Tools Studied during Internship**

**2.1 Java**

Java is a high-level programming language originally developed by Sun Microsystems and released in 1995. Java runs on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX. Some key advantages of Java Programming are:

**2.1.1 Advantages of Java**

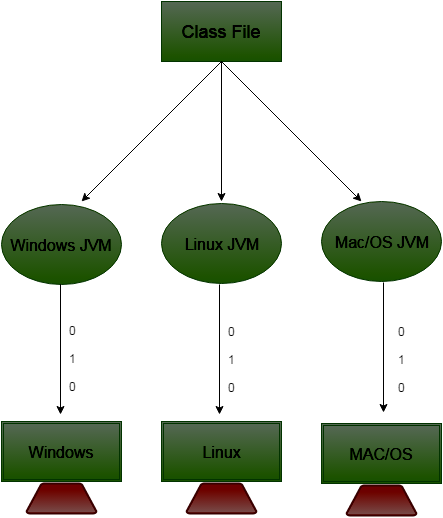
* **Object Oriented** − In Java, everything is an Object. Java can be easily extended since it is based on the Object model.



**(Fig 2.1: Object Oriented Features of Java)**

* **Platform Independent** − Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine, rather into platform independent byte code. This byte code is distributed over the web and interpreted

by the Virtual Machine (JVM) on whichever platform it is being run on.

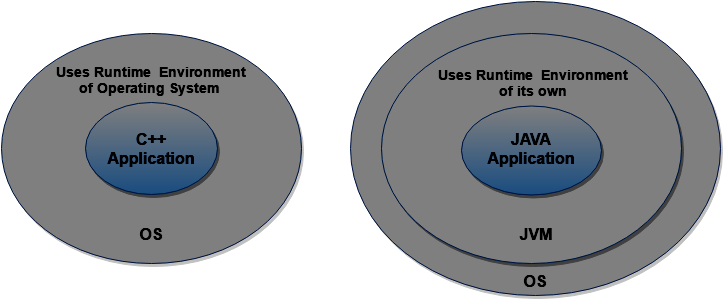


**(Fig 2.2: JVM Platform Independence)**

* **Simple** − Java is designed to be easy to learn. If you understand the basic concept of OOP Java, it would be easy to master.
* **Secure** − With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.

Also, Java is secure because:

* 1. Java does not have any explicit pointer.
  2. Java programs run inside virtual machine sandbox
  3. **Class Loader:** It adds security by separating the package for the classes of the local file system from those that are imported from the network.

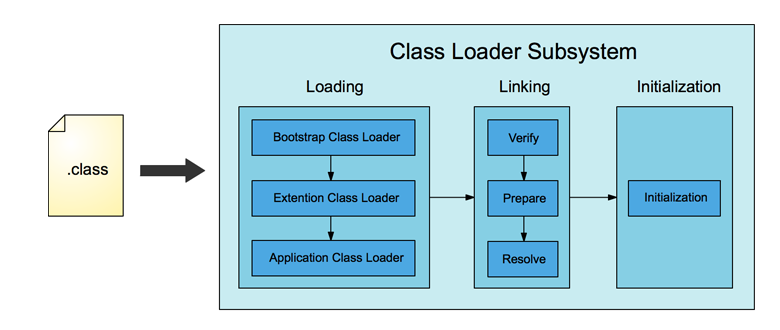


**(Fig 2.3: Security In Java due to Virtual Machine Sandbox)**

Actually, the Java Virtual Machine (JVM) has the class loader which loads all the Java application, run-time and Java Standard Edition (SE) classes to the main memory for the execution of a Java program. After the loading stage, there is a linking stage where the JVM verifier verifies the bytecode generated by the Java Compiler.

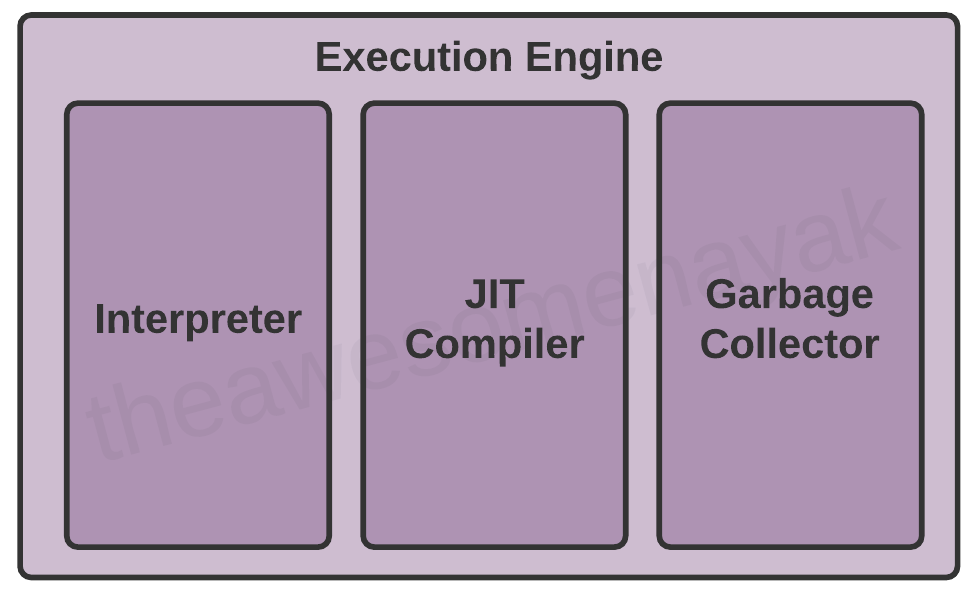
If there is any mismatch in what the original bytecode pattern should have been, this means that there is some malware in the Java bytecode and we will get a verification exception thrown.

Thus, such a program with malwares is not executed and we get a secure system in Java.



**(Fig 2.4: Verify Procedure by JVM verifier provides Security)**

* 1. **Security Manager:** It determines what resources a clad can access such as reading and writing to a local disk.
* **Architecture-neutral** − Java compiler generates an architecture-neutral object file format, which makes the compiled code executable on many processors, with the presence of Java runtime system. Though, the most commonly used architecture is the Von-Neumann Architecture, still Java can run on any other architecture as well.
* **Portable** − Being architecture-neutral and having no implementation dependent aspects of the specification makes Java portable. Compiler in Java is written in ANSI C with a clean portability boundary, which is a POSIX subset.
* **Robust** − Java makes an effort to eliminate error prone situations by emphasizing mainly on compile time error checking and runtime checking.
* **Dynamic Memory Based System**: Java has a dynamic memory-based system i.e. most of the components of a Java program such as an array or a string or the user defined objects are all created inside heap memory section. This allows the developer the freedom of not worrying about the limited stack space and also of thinking freely at the time of creation of arrays, strings and other objects and not worrying that which components or objects or variables should be there in stack and which should be there in heap.
* **Java Execution Engine:** The java execution engine comprises of three components and each one of these components has their own benefits. The three components are **Interpreter**, **Just In Time (JIT) compiler** and **Garbage Collector**.

****

**(Fig 2.5: Execution Engine in JVM)**

**Interpreter:** The work of the interpreter is to execute the bytecode generated by the Java compiler line-by-line.

**Just-In-Time (JIT) Compiler:** Let us say that there are a few repeating lines in the code or let’s say that there is a loop running for 10 thousand times in the java code. Now, the same lines of code within that loop should not be interpreted 10 thousand times. So, JIT compiler takes care of these repeating lines of code to improve the performance in Java. The JIT compiler converts the certain piece of repeating code into machine code and then executes it repeatedly without interpreting it every time.

**Garbage Collector:** The garbage collector is another very important feature in Java. In languages like C/C++, we use pointers and we know that the pointers can cause a lot of problems. For instance, if an object i.e. the instance has been deleted but the reference i.e. the pointer to it still exists, the pointer will point to some random memory location and this can cause a serious crash of application and memory can also get corrupted. Since there are no pointers in Java, this type of situation is uncommon but it may happen that the objects that are not in use in the program might still occupy the memory. So, to free that memory, garbage collector deletes the objects from the heap that are no more in use in the program.

**2.1.2 Applications of Java Programming:**

* **Multithreaded** − With Java's multithreaded feature it is possible to write programs that can perform many tasks simultaneously. This design feature allows the developers to construct interactive applications that can run smoothly.



**(Fig 2.6: Advantages of Multithreading in Java)**

* **Interpreted** − Java byte code is translated on the fly to native machine instructions and is not stored anywhere. The development process is more rapid and analytical since the linking is an incremental and light-weight process.
* **High Performance** − With the use of Just-In-Time compilers, Java enables high performance.
* **Distributed** − Java is designed for the distributed environment of the internet.
* **Dynamic** − Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment. Java programs can carry extensive amount of run-time information that can be used to verify and resolve accesses to objects on run-time.

**2.2 Java Swing**

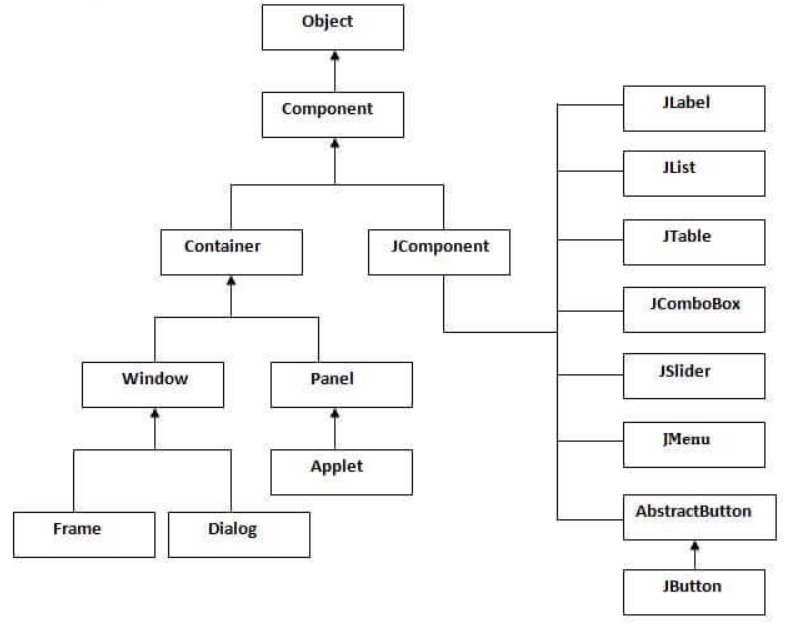
Java Swing is a frameworkthat is *used to create window-based applications*. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java. Unlike AWT, Java Swing provides platform-independent and lightweight components.

The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

**2.2.1 Difference between Java Swing and AWT (Abstract Window Toolkit)**

* AWT components are platform dependent whereas the Java Swing components are platform independent.
* AWT components are heavy weight whereas the Swing components are light weight.
* AWT does not support pluggable look and feel whereas Swing does.
* AWT provides less components than Swing. Swing provides more powerful components such as tables, lists, scroll panes, color chooser, tabbed panes etc.
* AWT does not follow MVC (Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view. On the other hand, Swing follows MVC.

There are various Java Swing classes that are used to create static UI pages in Java applications. The hierarchy of the Java Swing classes are shown below:



**(Fig 2.7: Hierarchy of Java Swing Classes)**

**2.2.2 MVC Architecture**

Swing API architecture follows loosely based MVC architecture in the following manner:

* Model represents component’s data.
* View represents visual representation of the component’s data.
* Controller takes the input from the user on the view and reflects the changes in the Component’s data.
* Swing component has model as a separate element, while the view and the controller part are clubbed in the User Interface elements because of which Swing has a pluggable look and feel architecture.

**2.2.3 Swing Features:**

* **Light Weight:** Swing components are independent of native Operating System’s API as Swing API controls are rendered mostly using pure Java code instead of underlying operating system’s calls.
* **Rich Controls:** Swing provides a rich set of advanced controls like Tree, TabbedPane, slider, colorpicker and table controls.
* **Highly Customizable:** Swing controls can be customized in a very easy way as visual appearance is independent of internal representation.
* **Pluggable look-and-feel**: Swing based GUI application look and feel can be changed at run time, based on available values.

**Chapter-3**

**Demonstration of Technology through Project**

**Problem Description and objectives**

**3.1 Problem Description**

Given a collection of data records, we want to create a B+ tree index on some key field.

1. One approach is to insert each record into an empty tree. It is less expensive as compared to the indexing of that key field because each entry in the B/B+ tree requires us to start from the root and go down to the appropriate leaf page.
2. An efficient alternative to indexing is to use B/B+ tree. Hence, we want to implement a B+ tree and compare its performance with normal l indexing, with different inputs and document the results.

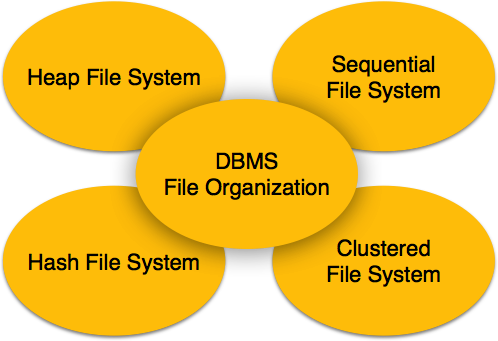
**3.2 Objective**

1. Implement a Table Scan for the relational database management System.
2. Implement Index Seek for Relational Database Management System .
3. Compare the efficiency of Index Seek and Table Scan in the relational database.
4. Implement Multilevel Indexing in Relational Database Management System
5. Implement B+Tree for better query optimization
6. Better search in less time.
7. Enhancing user efficiency.
8. Creates a connection by making it easy for others to plug into the platform.

**DBMS File Organisation**

Relative data and information are stored collectively in file formats. A file is a sequence of records stored in binary format. A disk drive is formatted into several blocks that can store records. File records are mapped onto those disk blocks.

File Organization defines how file records are mapped onto disk blocks. We have four types of File Organization to organize file records −



**(fig 3.1 Database File System)**

**3.3Heap File Organization**

When a file is created using Heap File Organization, the Operating System allocates memory area to that file without any further accounting details. File records can be placed anywhere in that memory area. It is the responsibility of the software to manage the records. Heap File does not support any ordering, sequencing, or indexing on its own.

**3.4 Sequential File Organization**

Every file record contains a data field (attribute) to uniquely identify that record. In a sequential file organization, records are placed in the file in some sequential order based on the unique key field or search key. Practically, it is not possible to store all the records sequentially in physical form.

**3.5 Hash File Organization**

Hash File Organization uses Hash function computation on some fields of the records. The output of the hash function determines the location of the disk block where the records are to be placed.

**3.6 Clustered File Organization**

Clustered file organization is not considered good for large databases. In this mechanism, related records from one or more relations are kept in the same disk block, that is, the ordering of records is not based on primary key or search key.

**3.7 File Operations**

Operations on database files can be broadly classified into two categories −

***3.7.1Update Operations***

**3*.7.2Retrieval Operations***

Update operations change the data values by insertion, deletion, or update. Retrieval operations, on the other hand, do not alter the data but retrieve them after optional conditional filtering. In both types of operations, selection plays a significant role. Other than the creation and deletion of a file, there could be several operations, which can be done on files.

**Open** − A file can be opened in one of the two modes, **read mode** or **write mode**. In read mode, the operating system does not allow anyone to alter data. In other words, data is read-only. Files opened in reading mode can be shared among several entities. Write mode allows data modification. Files opened in write mode can be read but cannot be shared.

* **Locate** − Every file has a file pointer, which tells the current position where the data is to be read or written. This pointer can be adjusted accordingly. Using find (seek) operation, it can be moved forward or backward.
* **Read** − By default, when files are opened in reading mode, the file pointer points to the beginning of the file. There are options where the user can tell the operating system where to locate the file pointer at the time of opening a file. The very next data to the file pointer is read.
* **Write** − User can select to open a file in write mode, which enables them to edit its contents. It can be deletion, insertion, or modification. The file pointer can be located at the time of opening or can be dynamically changed if the operating system allows doing so.
* **Close** − This is the most important operation from the operating system’s point of view. When a request to close a file is generated, the operating system
  + removes all the locks (if in shared mode),
  + saves the data (if altered) to the secondary storage media, and
  + releases all the buffers and file handlers associated with the file.

The organization of data inside a file plays a major role here. The process to locate the file pointer to the desired record inside a file various based on whether the records are arranged sequentially or clustered.

**3.8 DBMS Data Storage System**

Databases are stored in file formats, which contain records. At the physical level, the actual data is stored in an electromagnetic format on some device. These storage devices can be broadly categorized into three types –



**(Fig 3.2: Memory Hierarchy)**

***3.8.1Primary Storage*** − The memory storage that is directly accessible to the CPU comes under this category. CPU's internal memory (registers), fast memory (cache), and main memory (RAM) are directly accessible to the CPU, as they are all placed on the motherboard or CPU chipset. This storage is typically very small, ultra-fast, and volatile. Primary storage requires a continuous power supply in order to maintain its state. In case of a power failure, all its data is lost.

***3.8.2Secondary Storage*** − Secondary storage devices are used to store data for future use or as a backup. Secondary storage includes memory devices that are not a part of the CPU chipset or motherboard, for example, magnetic disks, optical disks (DVD, CD, etc.), hard disks, flash drives, and magnetic tapes.

***3.8.3Tertiary Storage*** − Tertiary storage is used to store huge volumes of data. Since such storage devices are external to the computer system, they are the slowest in speed. These storage devices are mostly used to take the back up of an entire system. Optical disks and magnetic tapes are widely used as tertiary storage.

**Study of Indexing**

**3.9 Indexing**

We know that data is stored in the form of records. Every record has a key field, which helps it to be recognized uniquely.

Indexing is a storage structure technique to efficiently retrieve records from the database files based on some attributes on which the indexing has been done. Indexing in database systems is similar to what we see in books.

Indexing is defined based on its indexing attributes. Indexing can be of the following types −

* PrimaryIndex − Primary index is defined on an ordered data file. The data file is ordered on a key field. The key field is generally the primary key of the relation.
* Secondary Index − Secondary index may be generated from a field which is a candidate key and has a unique value in every record, or a non-key with duplicate values.
* Clustering Index − Clustering index is defined on an ordered data file. The data file is ordered on a non-key field.

Ordered Indexing is of two types −

* Dense Index
* Sparse Index

### *3.9.1Dense Index*

In the dense index, there is an index record for every search key value in the database. This makes searching faster but requires more space to store index records itself. Index records contain search key value and a pointer to the actual record on the disk.

## 

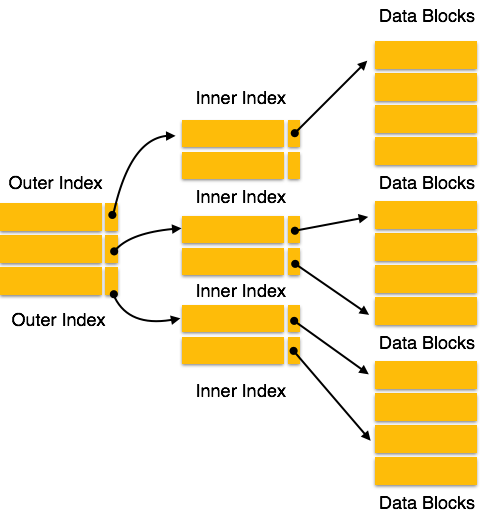
**(fig 3.3: Dense Indexing)**

### *3.9.2Sparse Index*

In the sparse index, index records are not created for every search key. An index record here contains a search key and an actual pointer to the data on the disk. To search a record, we first proceed by index record and reach the actual location of the data. If the data we are looking for is not where we directly reach by following the index, then the system starts the sequential search until the desired data is found.

### *3.9.3 Multilevel Index*

Index records comprise search-key values and data pointers. The multilevel index is stored on the disk along with the actual database files. As the size of the database grows, so does the size of the indices. There is an immense need to keep the index records in the main memory so as to speed up the search operations. If a single-level index is used, then a large size index cannot be kept in memory which leads to multiple disk accesses.



**(fig-3.4: Multilevel Indexing)**

Multi-level Index helps in breaking down the index into several smaller indices in order to make the outermost level so small that it can be saved in a single disk block, which can easily be accommodated anywhere in the main memory.

Index records comprise search-key values and data pointers. The multilevel index is stored on the disk along with the actual database files. As the size of the database grows, so does the size of the indices. There is an immense need to keep the index records in the main memory so as to speed up the search operations. If a single-level index is used, then a large size index cannot be kept in memory which leads to multiple disk accesses.

**B Tree**

**3.10 Introduction**

B Tree is a specialized m-way tree that can be widely used for disk access. A B-Tree of order m can have at most m-1 keys and m children. One of the main reason of using B tree is its capability to store large number of keys in a single node and large key values by keeping the height of the tree relatively small.

A B tree of order m contains all the properties of an M way tree. In addition, it contains the following properties.

1. Every node in a B-Tree contains at most m children.
2. Every node in a B-Tree except the root node and the leaf node contain at least m/2 children.
3. The root nodes must have at least 2 nodes.
4. All leaf nodes must be at the same level.

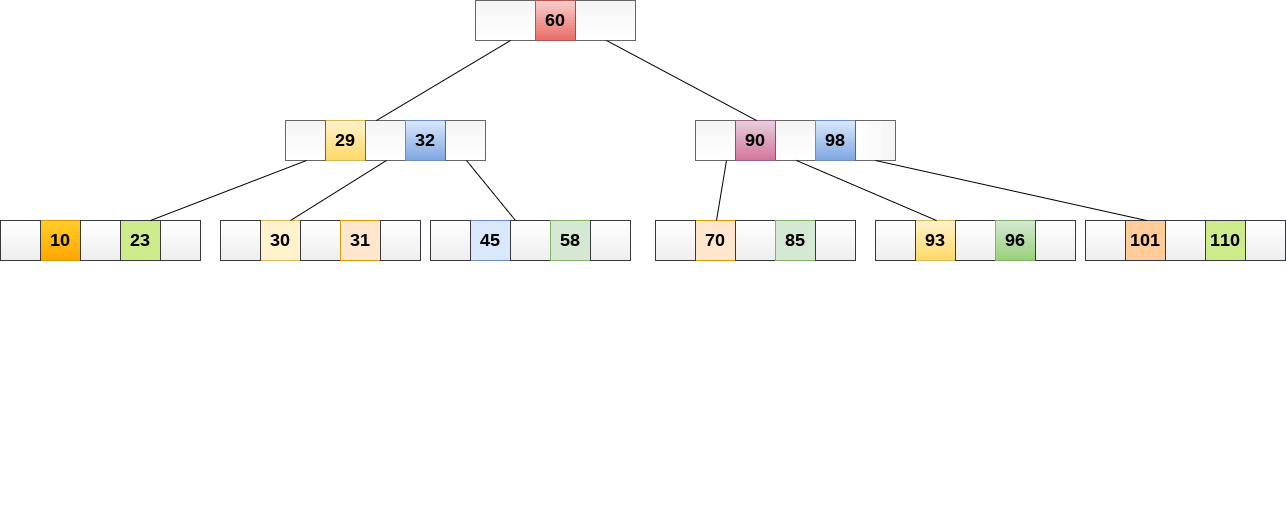
It is not necessary that, all the nodes contain the same number of children but, each node must have m/2 number of nodes.

A B tree of order 4 is shown in the following image.

13.7M

275

HTML Tutorial



**(fig-3.5: B -Tree)**

While performing some operations on B Tree, any property of B Tree may violate such as number of minimum children a node can have. To maintain the properties of B Tree, the tree may split or join.

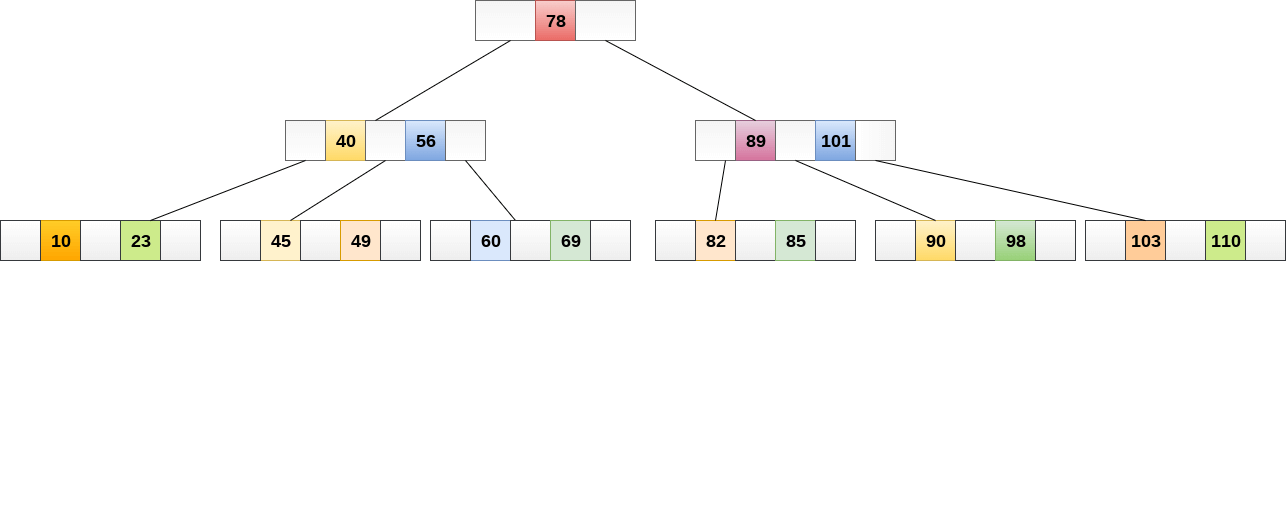
## 3.11 Operations

### 3.11.1 Searching:

Searching in B Trees is similar to that in Binary search tree. For example, if we search for an item 49 in the following B Tree. The process will something like following:

1. Compare item 49 with root node 78. since 49 < 78 hence, move to its left sub-tree.
2. Since, 40<49<56, traverse right sub-tree of 40.
3. 49>45, move to right. Compare 49.
4. match found, return.

Searching in a B tree depends upon the height of the tree. The search algorithm takes O(log n) time to search any element in a B tree.



### (fig-3.6: Searching in B-Tree)

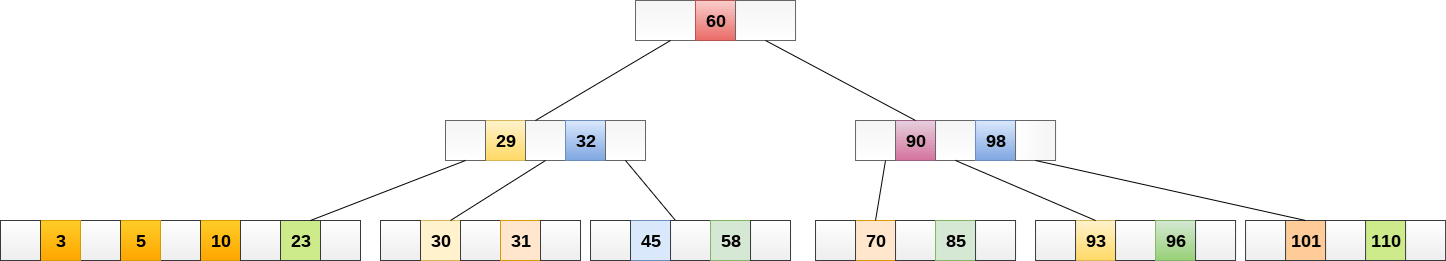
### 3.11.2 Inserting

Insertions are done at the leaf node level. The following algorithm needs to be followed in order to insert an item into B Tree.

1. Traverse the B Tree in order to find the appropriate leaf node at which the node can be inserted.
2. If the leaf node contain less than m-1 keys then insert the element in the increasing order.
3. Else, if the leaf node contains m-1 keys, then follow the following steps.
   * Insert the new element in the increasing order of elements.
   * Split the node into the two nodes at the median.
   * Push the median element upto its parent node.
   * If the parent node also contain m-1 number of keys, then split it too by following the same steps.

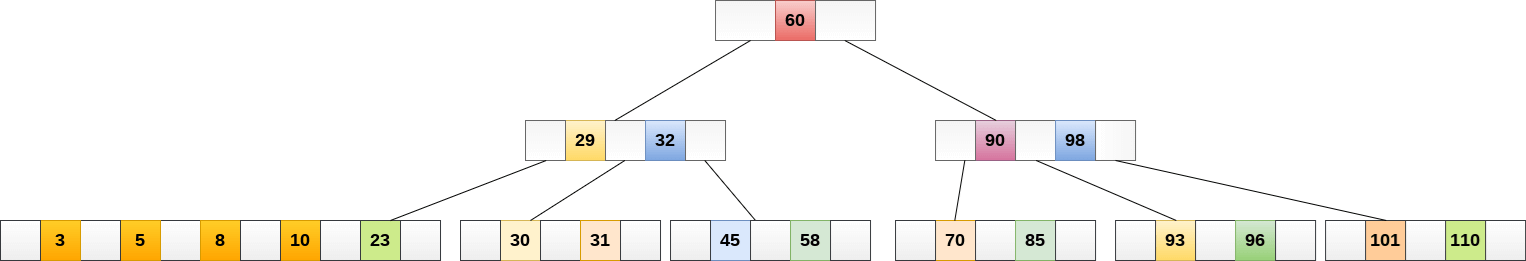
**Example:**

Insert the node 8 into the B Tree of order 5 shown in the following image.



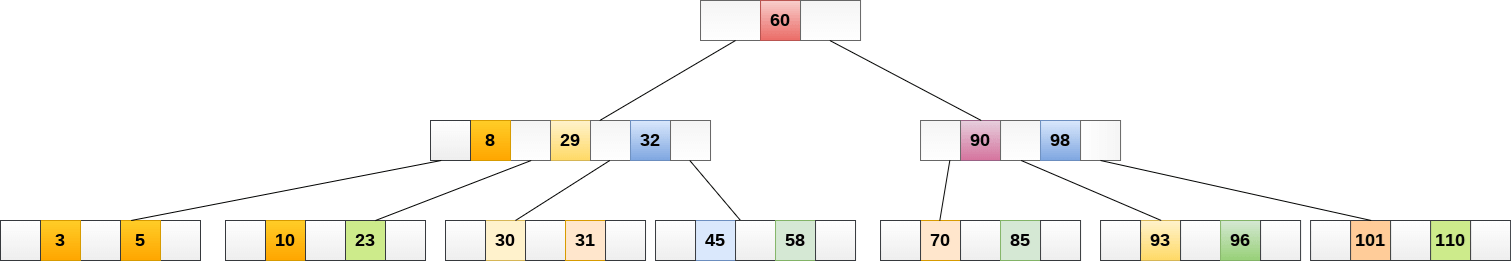
**(Fig-3.7: Inserting 8)**

8 will be inserted to the right of 5, therefore insert 8.



**(fig-3.8: B-Tree after inserting 8)**

The node, now contain 5 keys which is greater than (5 -1 = 4 ) keys. Therefore split the node from the median i.e. 8 and push it up to its parent node shown as follows.



**(fig-3.9: B-Tree after splitting of the Node)**

### 3.11.3 Deletion

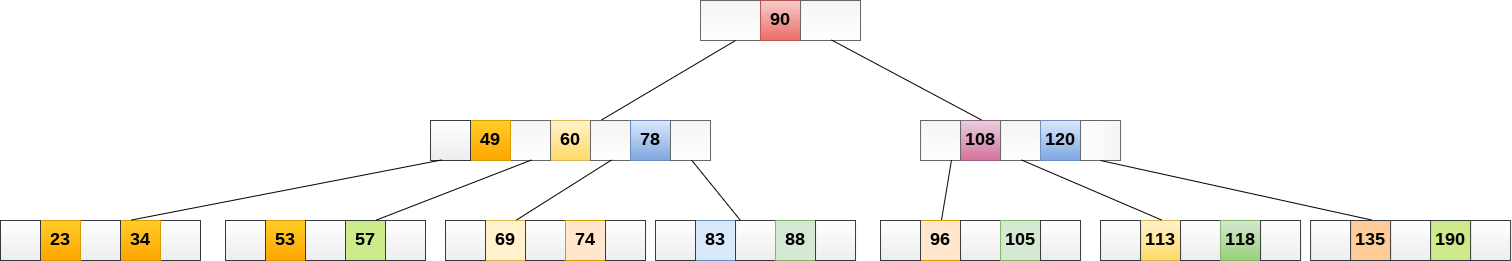
Deletion is also performed at the leaf nodes. The node which is to be deleted can either be a leaf node or an internal node. Following algorithm needs to be followed in order to delete a node from a B tree.

1. Locate the leaf node.
2. If there are more than m/2 keys in the leaf node then delete the desired key from the node.
3. If the leaf node doesn't contain m/2 keys then complete the keys by taking the element from eight or left sibling.
   * If the left sibling contains more than m/2 elements then push its largest element up to its parent and move the intervening element down to the node where the key is deleted.
   * If the right sibling contains more than m/2 elements then push its smallest element up to the parent and move intervening element down to the node where the key is deleted.
4. If neither of the sibling contain more than m/2 elements then create a new leaf node by joining two leaf nodes and the intervening element of the parent node.
5. If parent is left with less than m/2 nodes then, apply the above process on the parent too.

If the the node which is to be deleted is an internal node, then replace the node with its in-order successor or predecessor. Since, successor or predecessor will always be on the leaf node hence, the process will be similar as the node is being deleted from the leaf node.

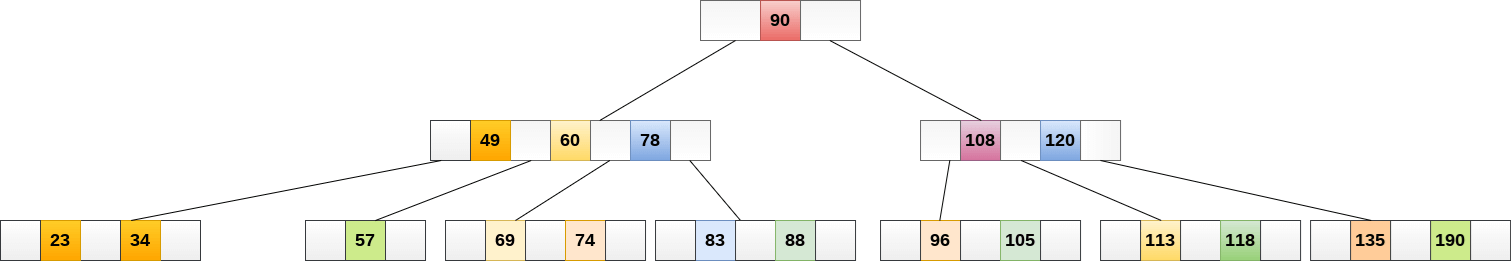
**Example 1**

Delete the node 53 from the B Tree of order 5 shown in the following figure.



**(fig-3.10: Delete Node 53)**

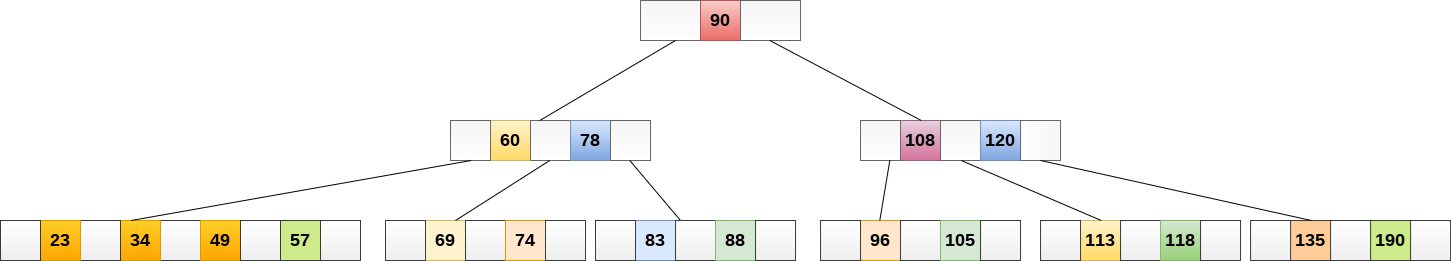
53 is present in the right child of element 49. Delete it.



**(fig:3.11- B-Tree after deleting Node 53)**

Now, 57 is the only element which is left in the node, the minimum number of elements that must be present in a B tree of order 5, is 2. It is less than that, the elements in its left and right sub-tree are also not sufficient therefore, merge it with the left sibling and intervening element of parent i.e. 49.

The final B tree is shown as follows.



**(fig-3.12: Final B-Tree after merging)**

## 3.12 Application of B tree

B tree is used to index the data and provides fast access to the actual data stored on the disks since, the access to value stored in a large database that is stored on a disk is a very time consuming process.

Searching an un-indexed and unsorted database containing n key values needs O(n) running time in worst case. However, if we use B Tree to index this database, it will be searched in O(log n) time in worst case.

**Implementation Details**

**3.13:** The leaves (the bottom-most index blocks) of the B+ tree are often linked to one another in a linked list; this makes range queries or an (ordered) iteration through the blocks simpler and more efficient (though the aforementioned upper bound can be achieved even without this addition). This does not substantially increase space consumption or maintenance on the tree. This illustrates one of the significant advantages of a B+ tree over a B-tree; in a B-tree, since not all keys are present in the leaves, such an ordered linked list cannot be constructed. A B+ tree is thus particularly useful as a database system index, where the data typically resides on disk, as it allows the B+ tree to actually provide an efficient structure for housing the data itself .

If a storage system has a block size of B bytes, and the keys to be stored have a size of k, arguably the most efficient B+ tree is one where although theoretically, the one-off is unnecessary, in practice there is often a little extra space taken up by the index blocks (for example, the linked list references in the leaf blocks). Having an index block which is slightly larger than the storage system's actual block represents a significant performance decrease; therefore, erring on the side of caution is preferable.

If nodes of the B+ tree are organized as arrays of elements, then it may take a considerable time to insert or delete an element as half of the array will need to be shifted on average. To overcome this problem, elements inside a node can be organized in a binary tree or a B+ tree instead of an array.

B+ trees can also be used for data stored in RAM. In this case, a reasonable choice for block size would be the size of the processor's cache line.

The space efficiency of B+ trees can be improved by using some compression techniques. One possibility is to use delta encoding to compress keys stored into each block. For internal blocks, space saving can be achieved by either compressing keys or pointers. For string keys, space can be saved by using the following technique: Normally the i-th entry of an internal block contains the first key of the block . Instead of storing the full key, we could store the shortest prefix of the first key of block that is strictly greater (in lexicographic order) than the last key of block i. There is also a simple way to compress pointers: if we suppose that some consecutive blocks are stored contiguously, then it will suffice to store only a pointer to the first block and the count of consecutive blocks.

All the above compression techniques have some drawbacks. First, a full block must be decompressed to extract a single element. One technique to overcome this problem is to divide each block into sub-blocks and compress them separately. In this case, searching or inserting an element will only need to decompress or compress a sub-block instead of a full block. Another drawback of compression technique is that the number of stored elements may vary considerably from a block to another depending on how well the elements are compressed inside each block.

### 3.14Implementation Blocks

***3.14.1 Insertion***  
1) Initialize x as root.  
2) While x is not a leaf, do the following  
a) Find the child of x that is going to to be traversed next. Let the child be y.  
b) If y is not full, change x to point toy.  
c) If y is full, split it and change x to point to one of the two parts of y. If k is smaller than mid key in y, then set x as the first part of y. The else second part of y. When we split y, we move a key from y to its parent x.  
3) The loop in step 2 stops when x is a leaf. x must have space for 1 extra key as we have been splitting all nodes in advance. So simply insert k to x.

***3.14.2 Searching***

1) Initialize x as root.  
2) While x is not a leaf, do the following:

a.) find the key value which equal to value to search, if found return the index.

b.) otherwise go to the node which may contain value to searched i.e. if the value to be searched is valid, then go to the node which has key values x<val<y.

c.) Recursively call the node.

3) The recursion in step 2 stops when x is leaf and we have not found value to be searched. In that case, we return key not found.

**3.14.3 Code Structure**

* For the implementation of B-Tree and creation of the random test data and index pages, there is only one file called the Data Manager which has all the implementation logic inside it.
* The Client File is used to fire the application i.e. The Client is the file that runs the Java Application by creating a constructor of App() class.
* The App file consists of all the Panels of the Application. The app constructor creates the Tabbed Pane and all the 4 Panels i.e. the Data Panel, The Home Panel, The Index Panel and the Query Panel.
* Each Panel has their own implementation of the private initComponents() method which sets the properties of the Panel such as the foreground, background colors and fonts and the text and elements alignment and so on.
* The Data Panel has an action Listener attached to the button so that the click fires the code for creating the test data with N number of rows entered by the user and 4 columns and those are Roll number, Name, User name and Password.
* Similarly, the index panel also has an action listener attached to the button for creating the index pages.
* The Query Panel has an action listener attached to the button to search the data according to the Table Scan or the Index Seek implementation.

**Chapter-4**

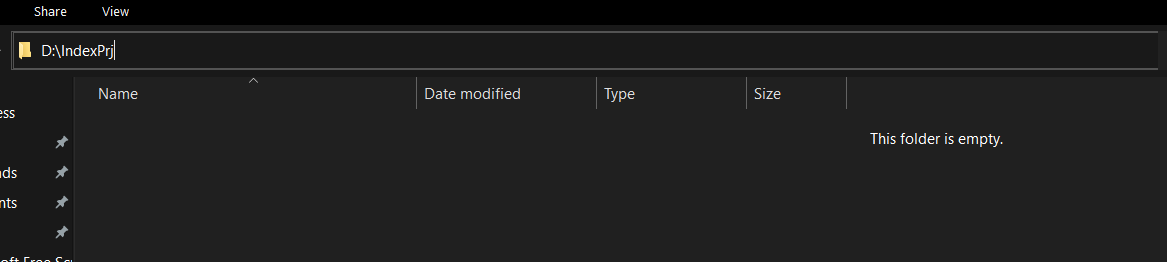
**Screenshots of the Project**

**4.1 Running the Project:** The project will be run by the Client java File which creates the App by calling the constructor.

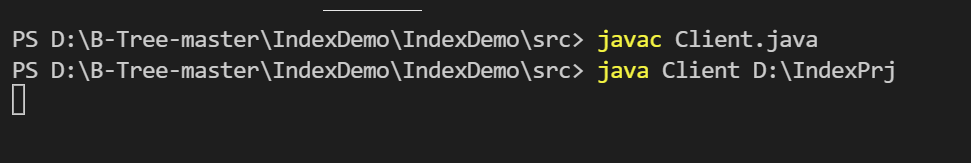


**(fig-4.1: Launches the app)**

**4.1.1 Command Line Argument:** As shown in the code snip above, we have to provide a directory path so that the database can be created in it. So, we create a folder named IndexPrj and we give its path as the command line argument while running the code.

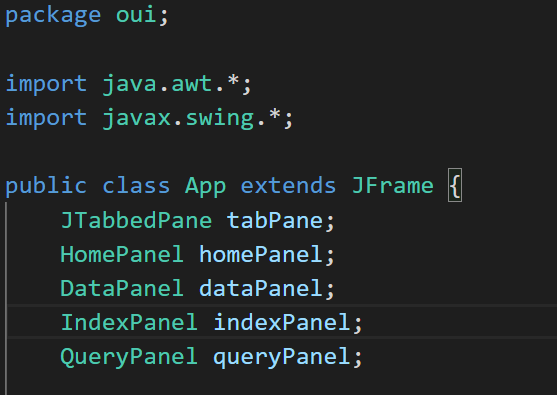


**(fig-4.2: Created a new Directory)**

****

**(fig-4.3: Gave its path as a Command Line argument)**

**4.1.2 App Launches:** The app launches after creating the folder and providing it as a command line argument to create the data base. App file has 4 panels namely HOME, DATA, INDICES and QUERY as shown below:



**(fig-4.4: Tabbed Panne has 4 panels)**

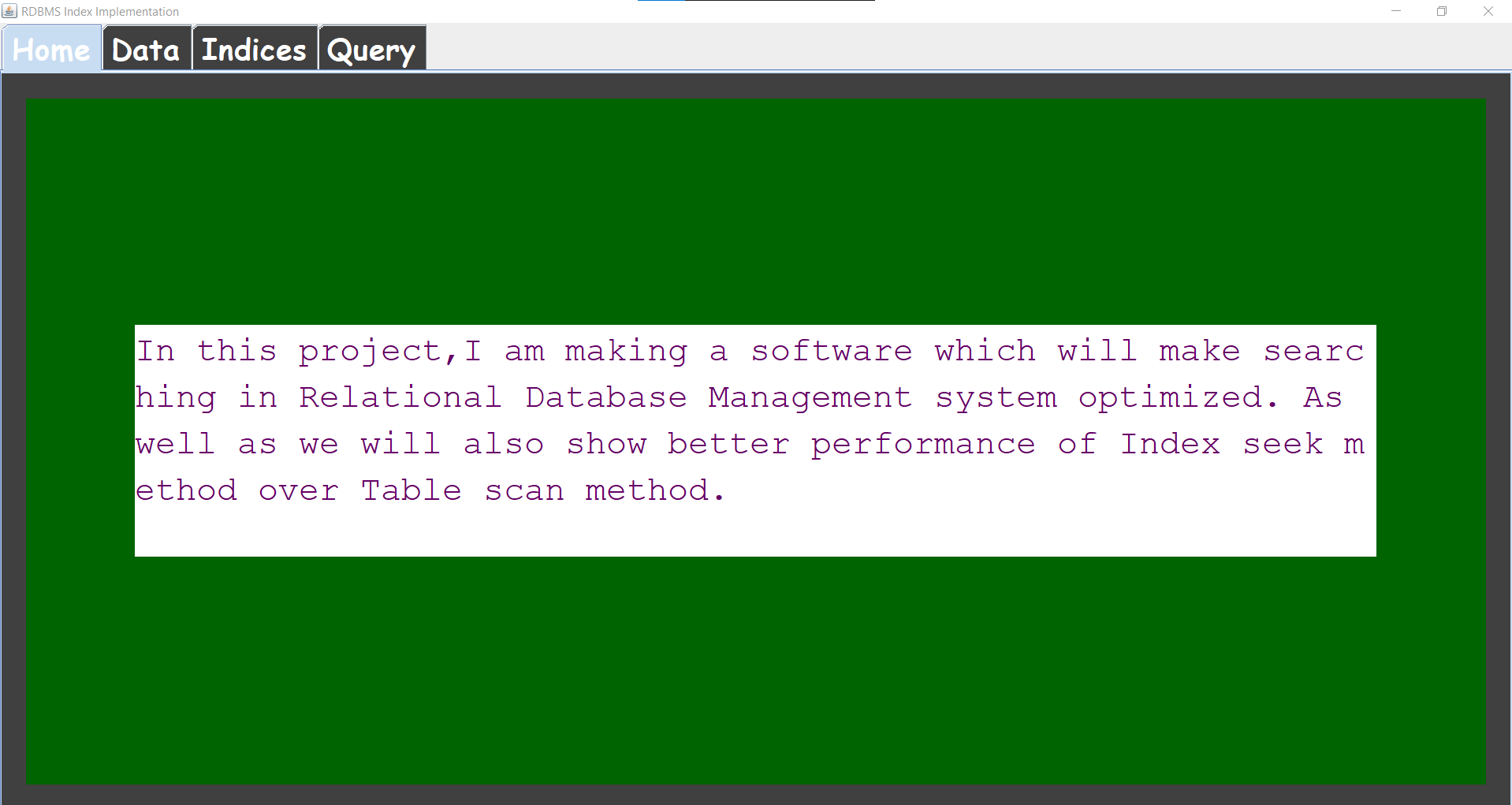
The app has all the four panels constructed and we have set the visibility to true so that the tab pane and the other components are visible. The tab pane was initially added to its super class i.e. JPane so that it appears on the window. The App() constructor is shown below:



**(fig-4.5: App constructor)**

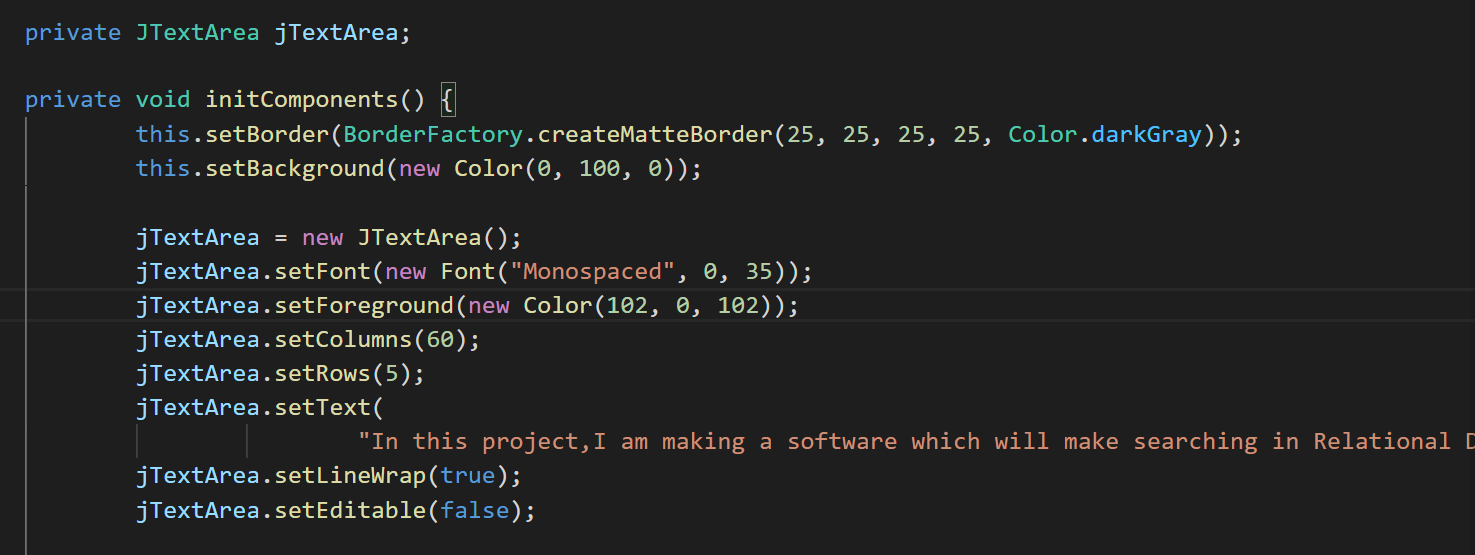
**4.2 Home Panel**:

When the app launches, we see the home panel i.e. the home screen which displays the motive of the project.



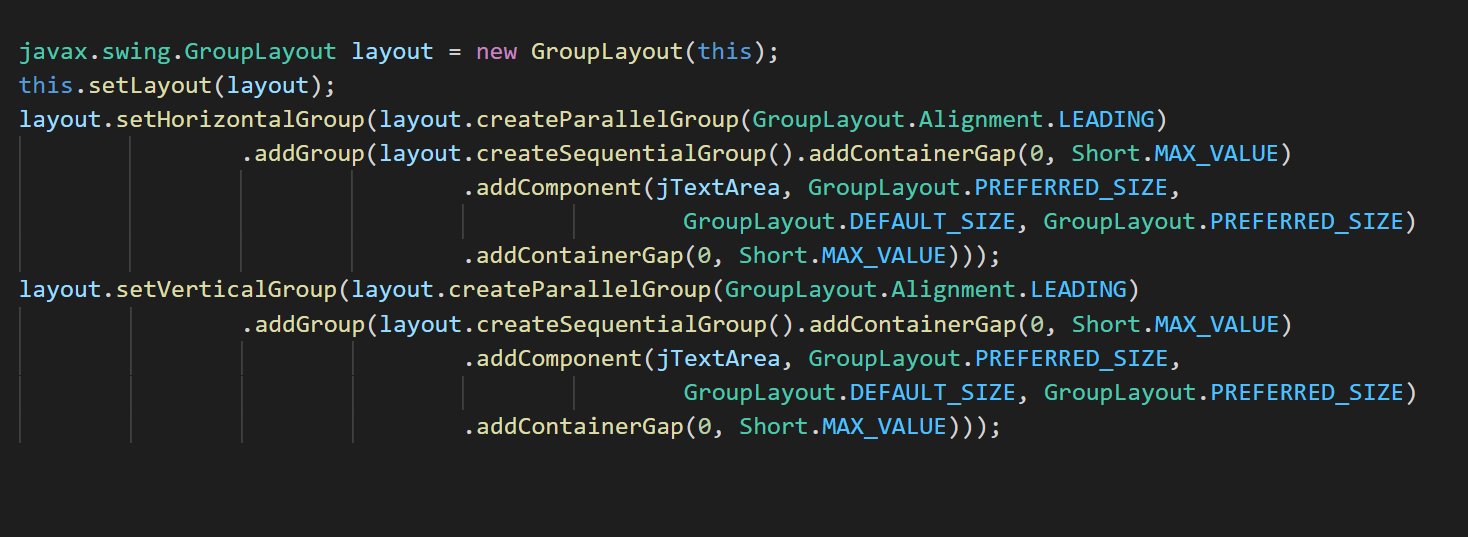
**(fig-4.6: Home Panel)**

We have set the foreground, background and other properties of the home panel in the HomePanel class private initComponents() method.



**(fig-4.7: Properties of HomePanel set in the initComponents() method)**

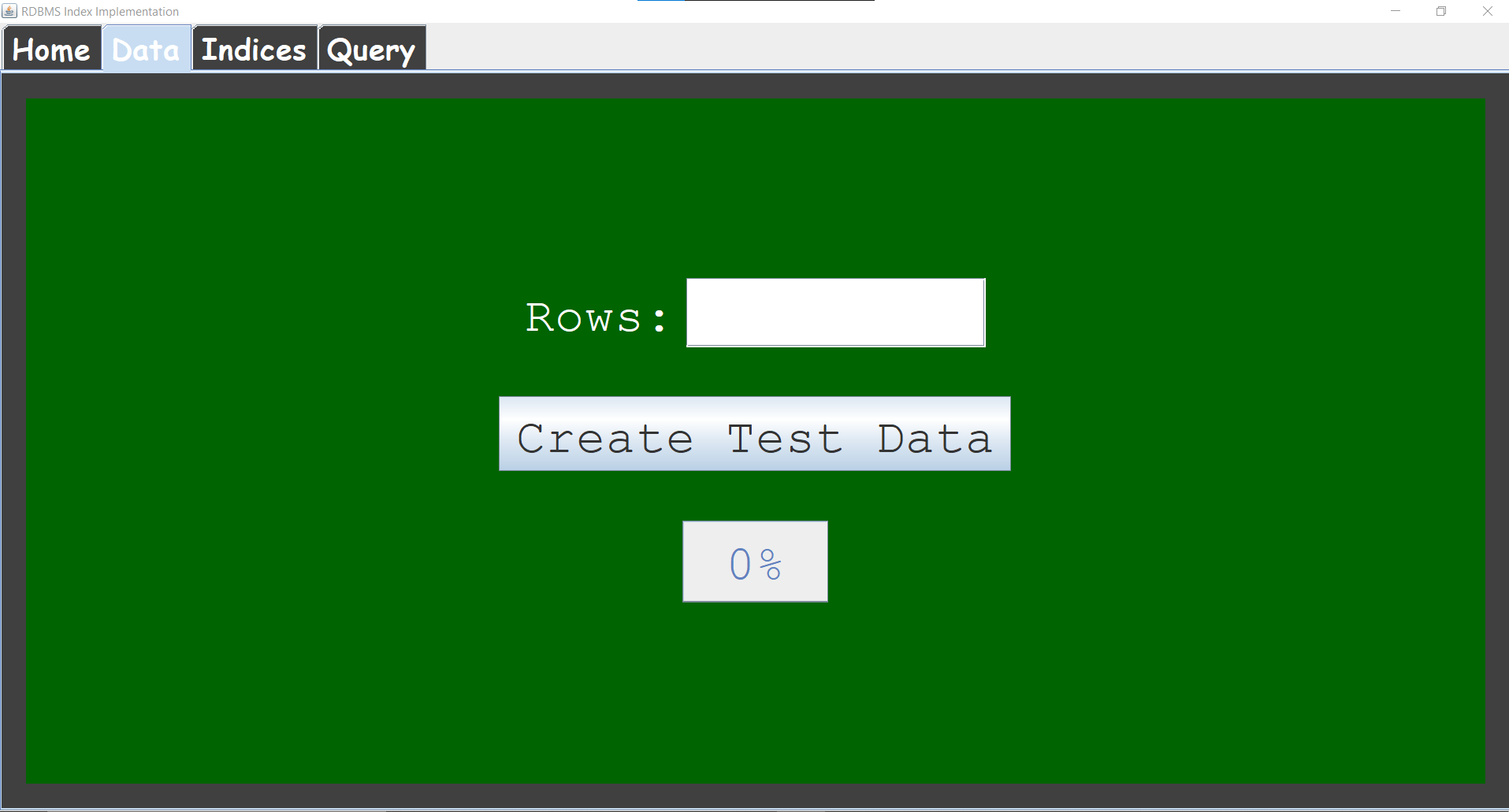
Then, we have set the alignment of the text using the group layout of Java Swing to the centre of the screen.



**(fig-4.8: Centre Alignment using Group Layout)**

**4.3 Data Panel:**

The data panel is where we can enter the number of rows to create some random data. The data has 4 columns and they are ID/Roll No, Name, Username and Password .The data panel is shown below:



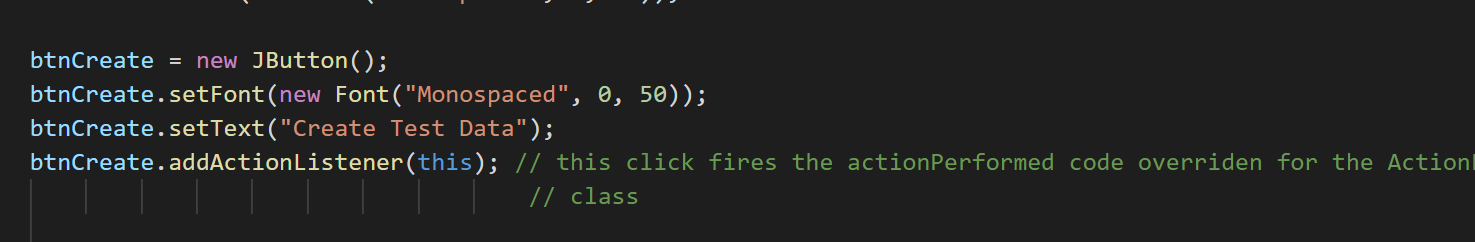
**(fig-4.9: Data Panel)**

The data panel has a label, a text field, a button and a progress bar. The label is “Rows” indicating that we have to enter the number of rows in the text field. The create Test Data button creates the random data with the number of rows specified and the number of columns as already discussed. The progress bar increases the progress percentage with the creation of data.



**(fig-4.10: The four components of the Data Panel)**

Again, the visibility and properties are all set in the initComponents() method of the Home Panel and it is a private method. When the Create Test Data button is clicked, the code according to the Obser-Design Pattern is designed such that the DataPanel listens to the action i.e. click on the button and fires the actionPerformed() thread.

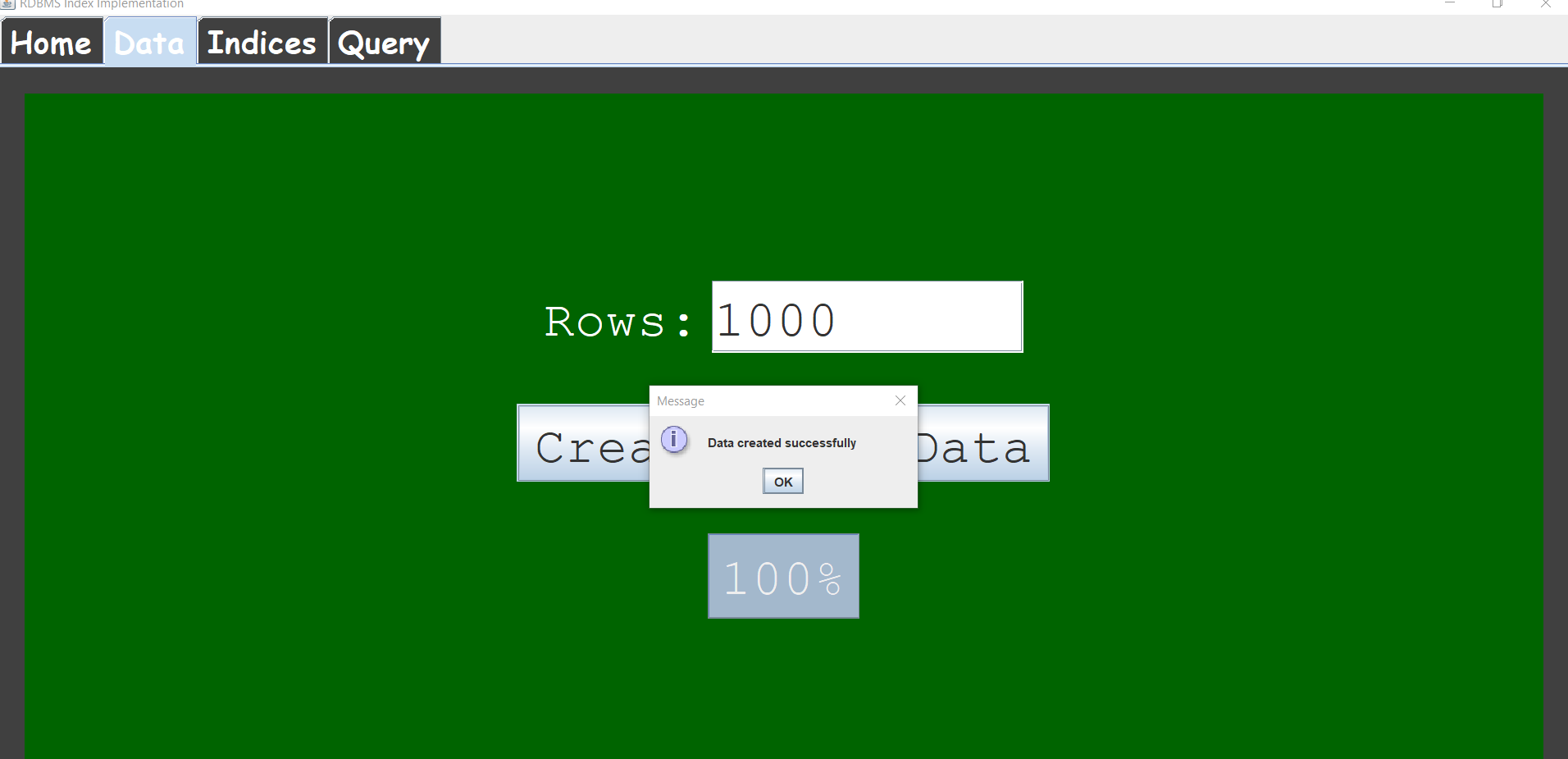


**(fig-4.11: Data Panel listens to the Click event on the button)**

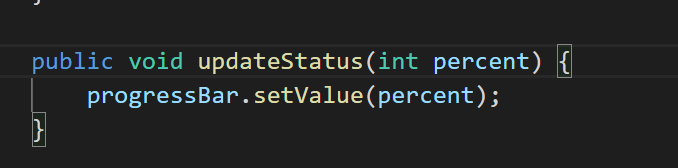
****

**(fig-4.12: The thread which creates test data is fires on the click)**

Also, the status of the progress bar updates as the data is created as shown below:

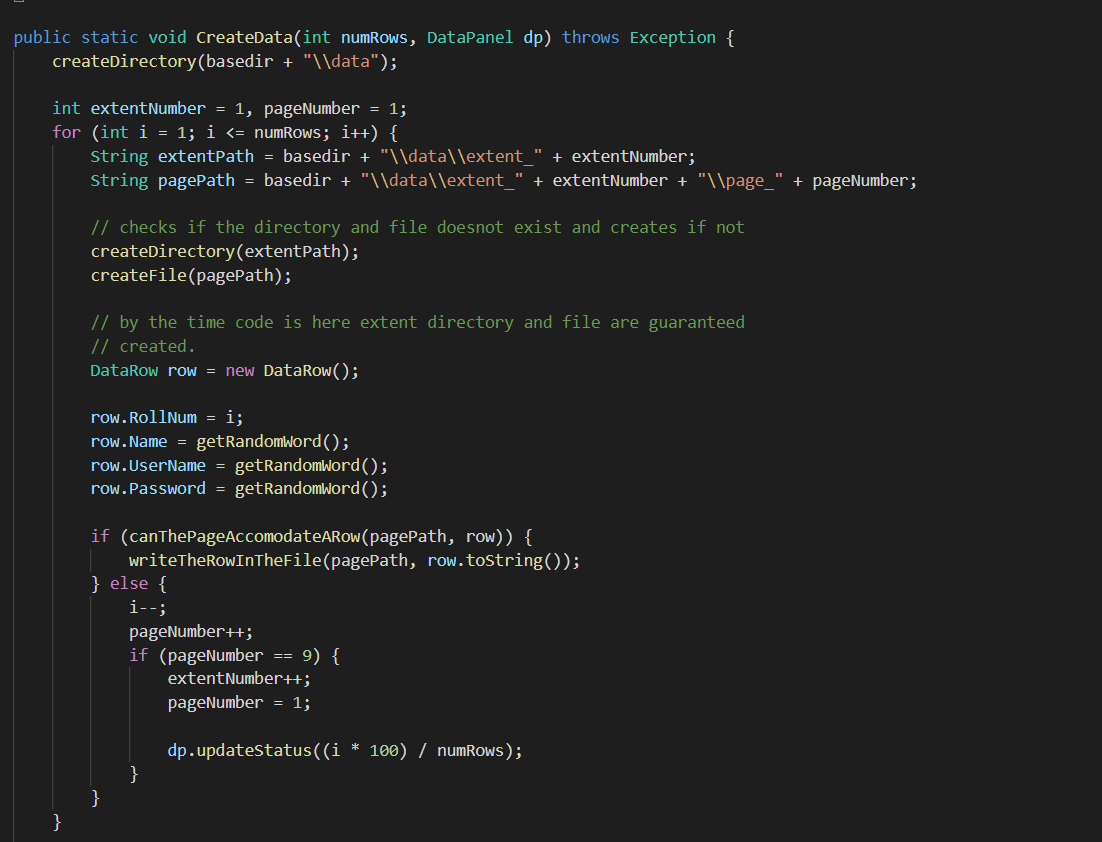


**(fig-4.13: Progress Bar updates and a message box appears when the data is created)**

****

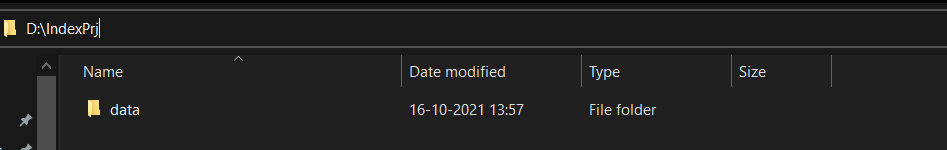
**(fig-4.14: This function updates the status of the progress bar)**

The entire logic for creating the test data and updating the progress bar is written inside the DataManager Class.



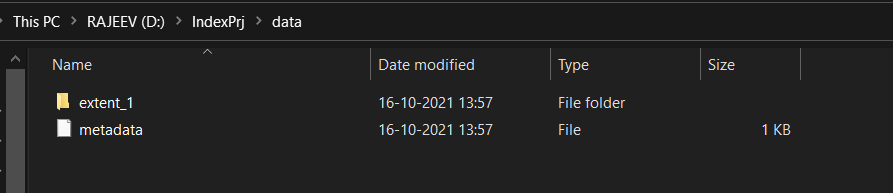
**(fig-4.15: Logic for data creation and updating the progress bar)**

Inside the directory that we created for the project to build a data base, we have a data folder created.

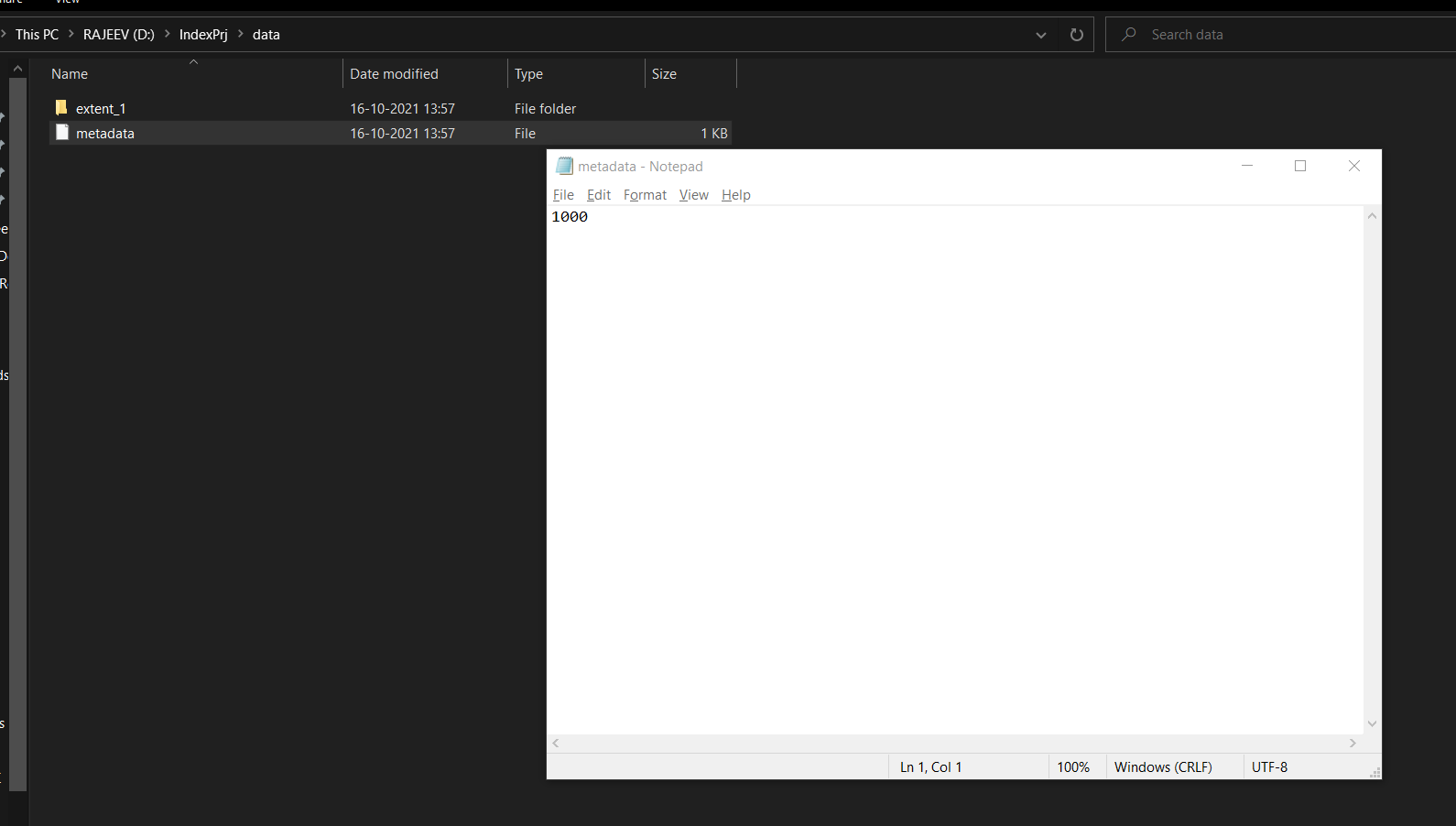


**(fig-4.16: Data Folder Created)**

Inside the data folder we have extents and the meta data file. The meta data file just stores the number of rows in the table.

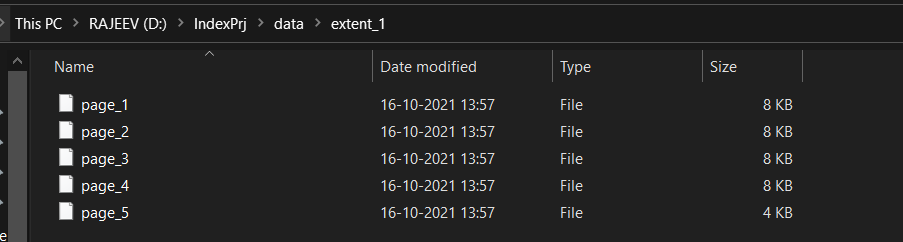


**(fig-4.17: Extents and Meta data inside the data folder)**

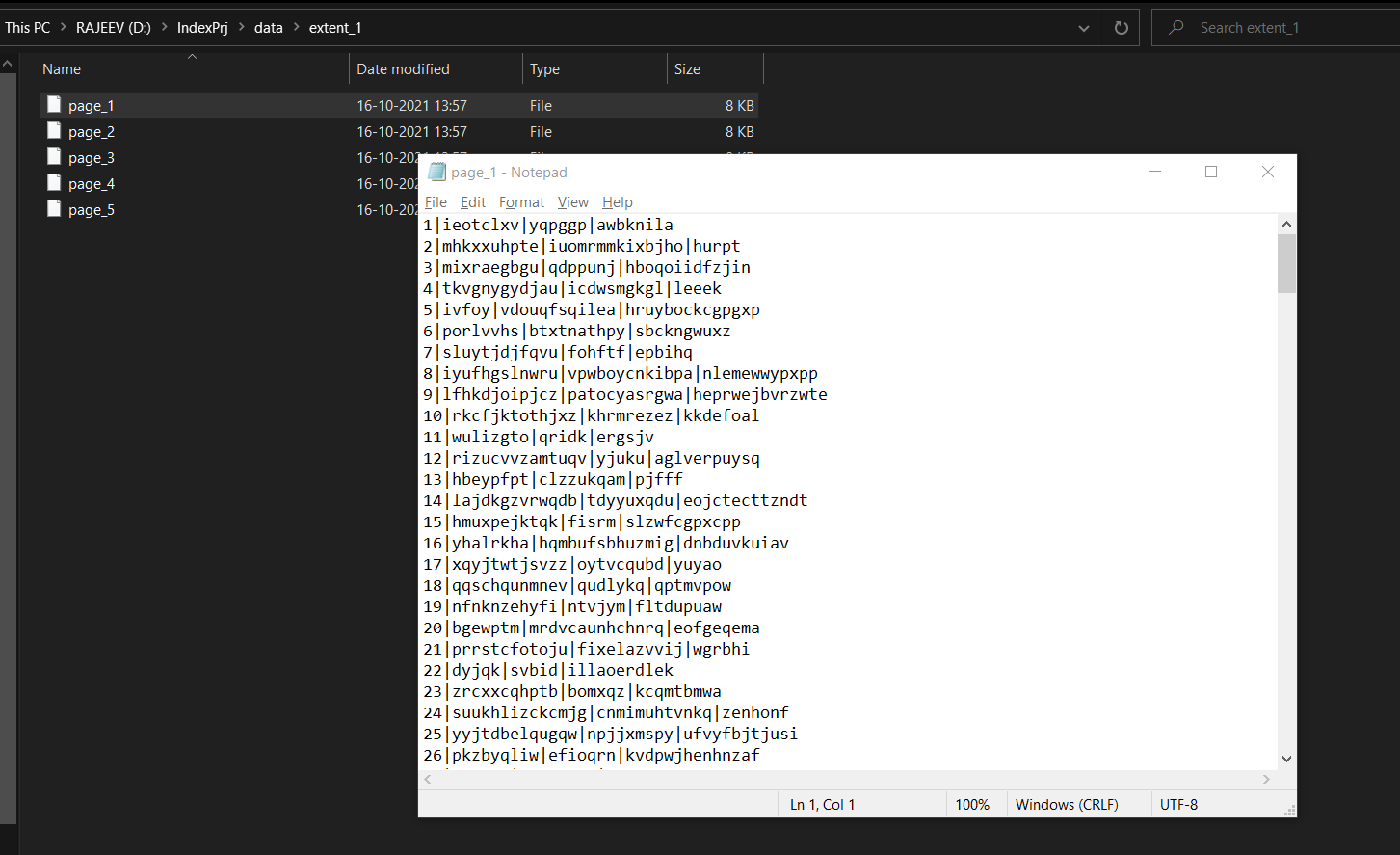
****

**(fig-4.18: Meta data file shows the number of rows)**

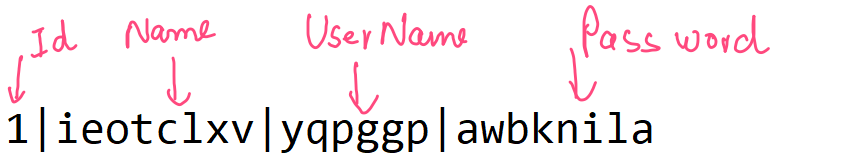
Inside the extents we have pages. The page size does not exceed 8 pages and each extent cannot have more than 8 pages.



**(fig-4.19: Pages inside extent and size does not exceed 8KB)**



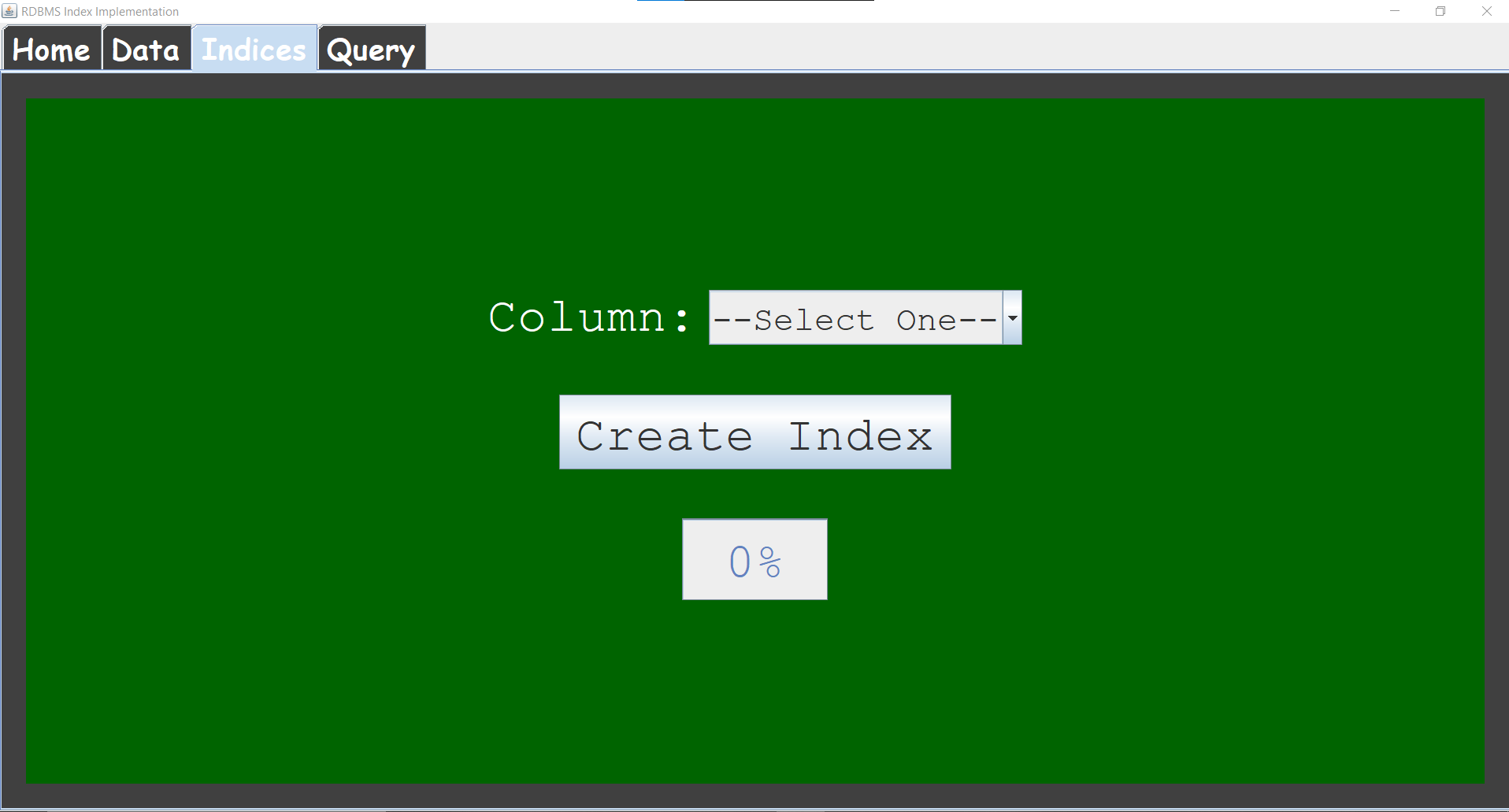
**(fig-4.20: Data inside of the pages)**



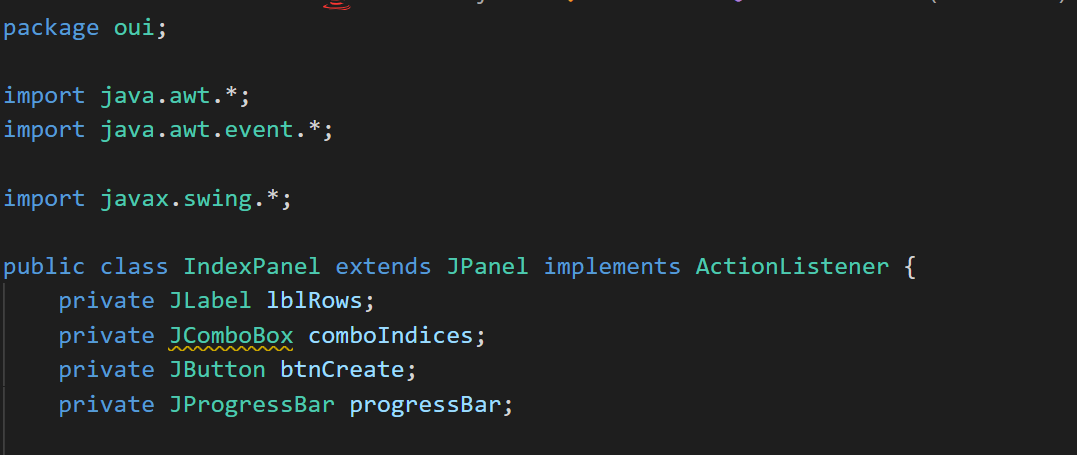
**(fig-4.21: Data inside each page divided into 4 columns)**

**4.4 Indices Panel:**

The Indices Panel has a label, a dropdown menu, a button and a progress bar as shown below:

****

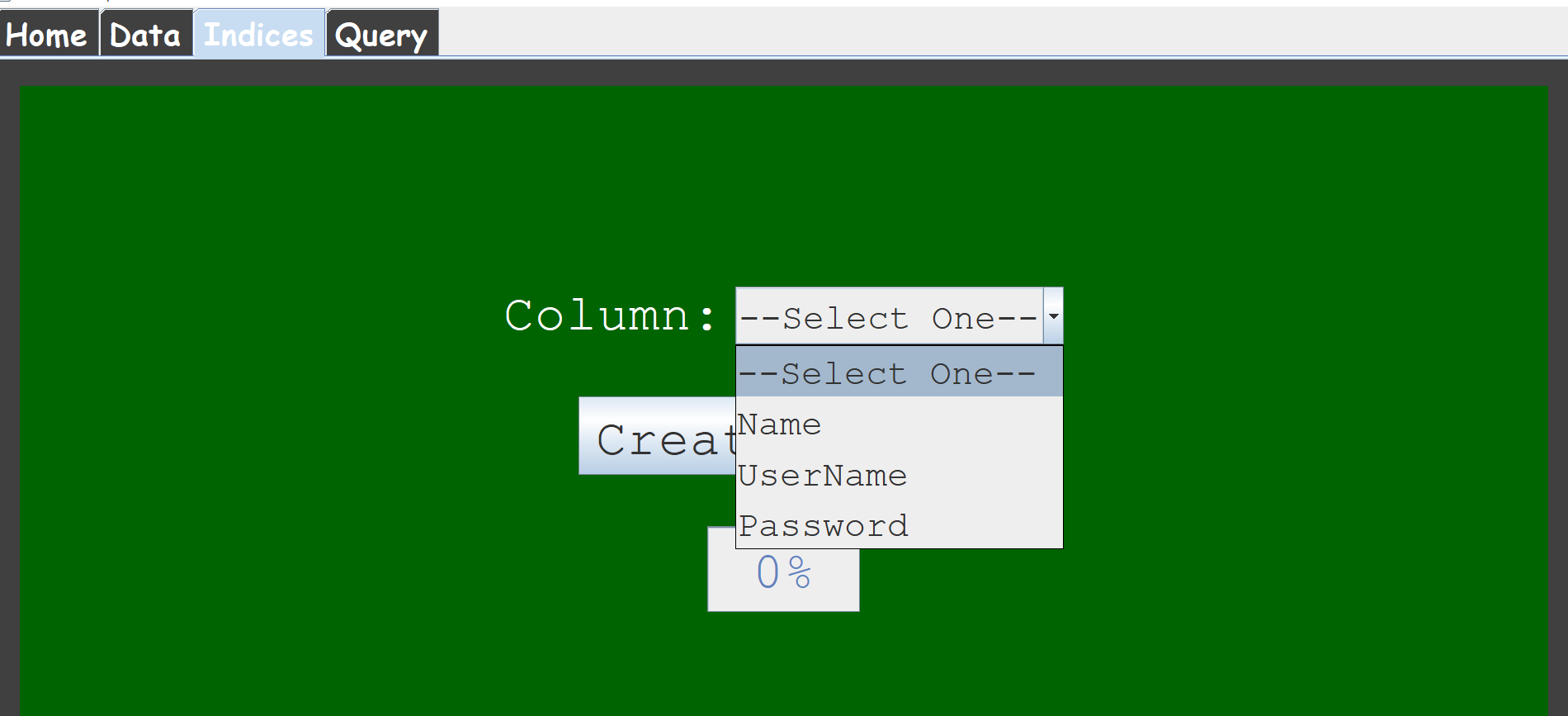
**(fig-4.22: The Indices Panel)**



**(fig-4.23: components of Indices Panel)**

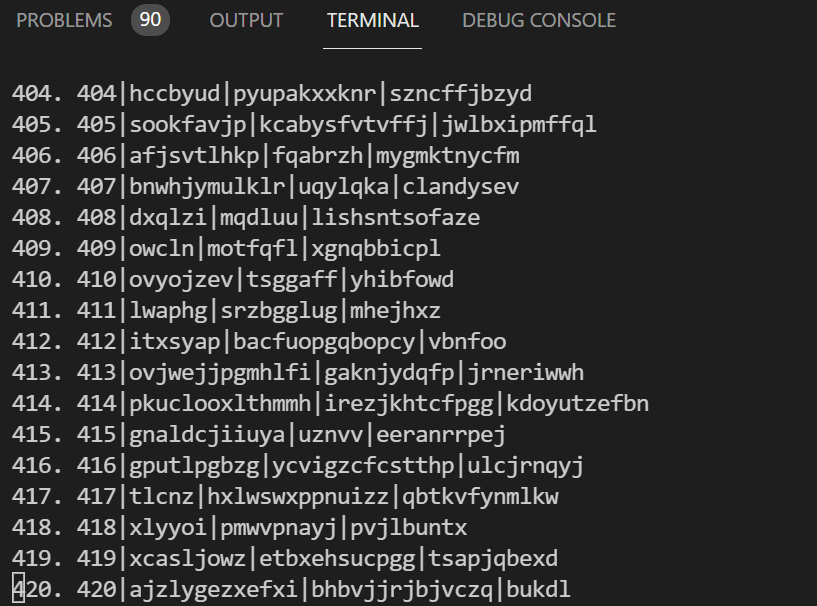
Like every other panel, the properties and visibility are set in the initComponents() private method of the IndexPanel class.

The dropdown menu lets us select from the three columns i.e. the Username, Name and password upon which we want to create the indices.

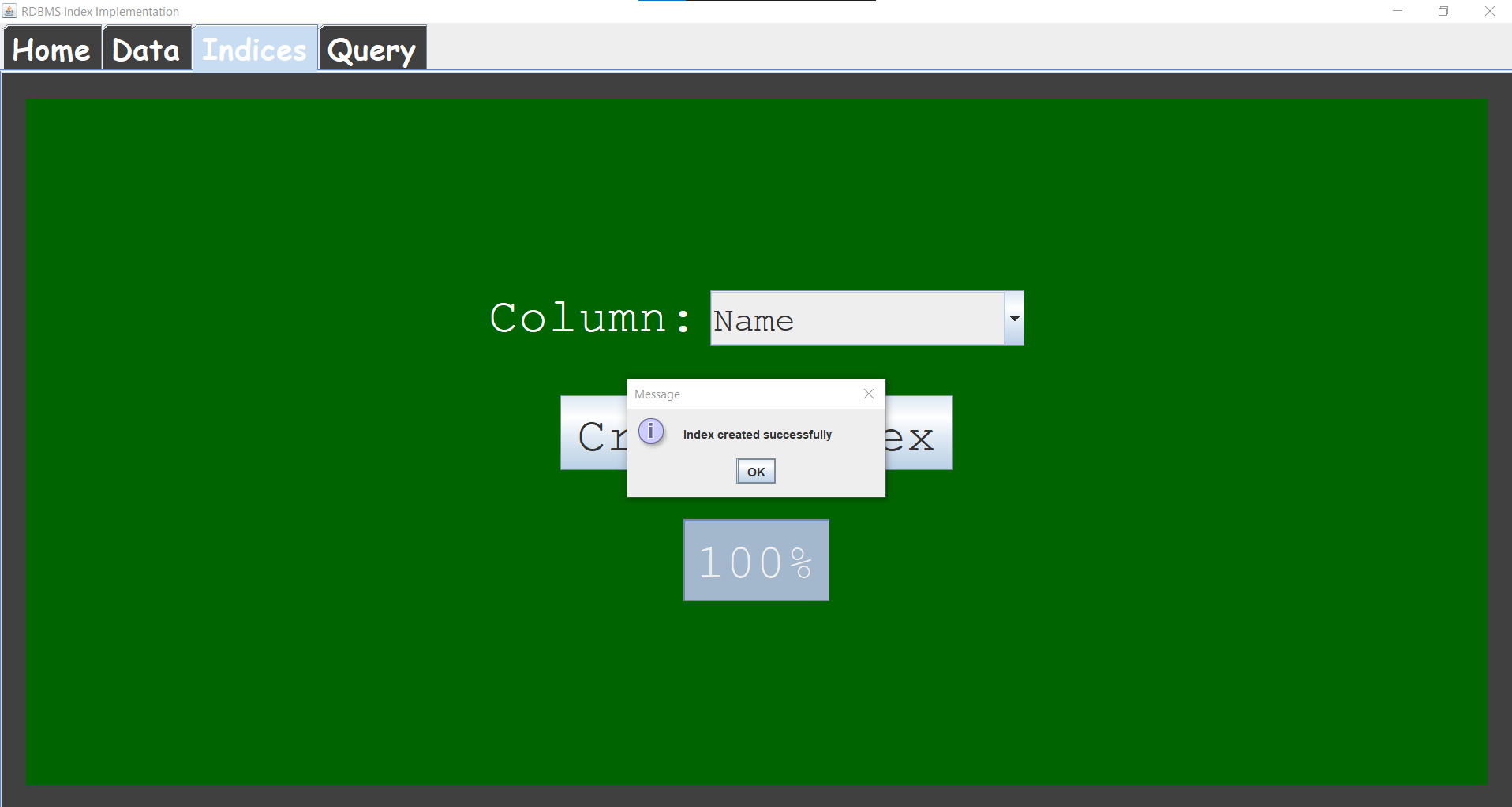


**(fig-4.24: Drop Down Options)**

Again, the IndexPanel will listen to the action performed on the Create Index button and create the indices for the field selected in the drop down menu.. The progress bar will also update accordingly as shown below:

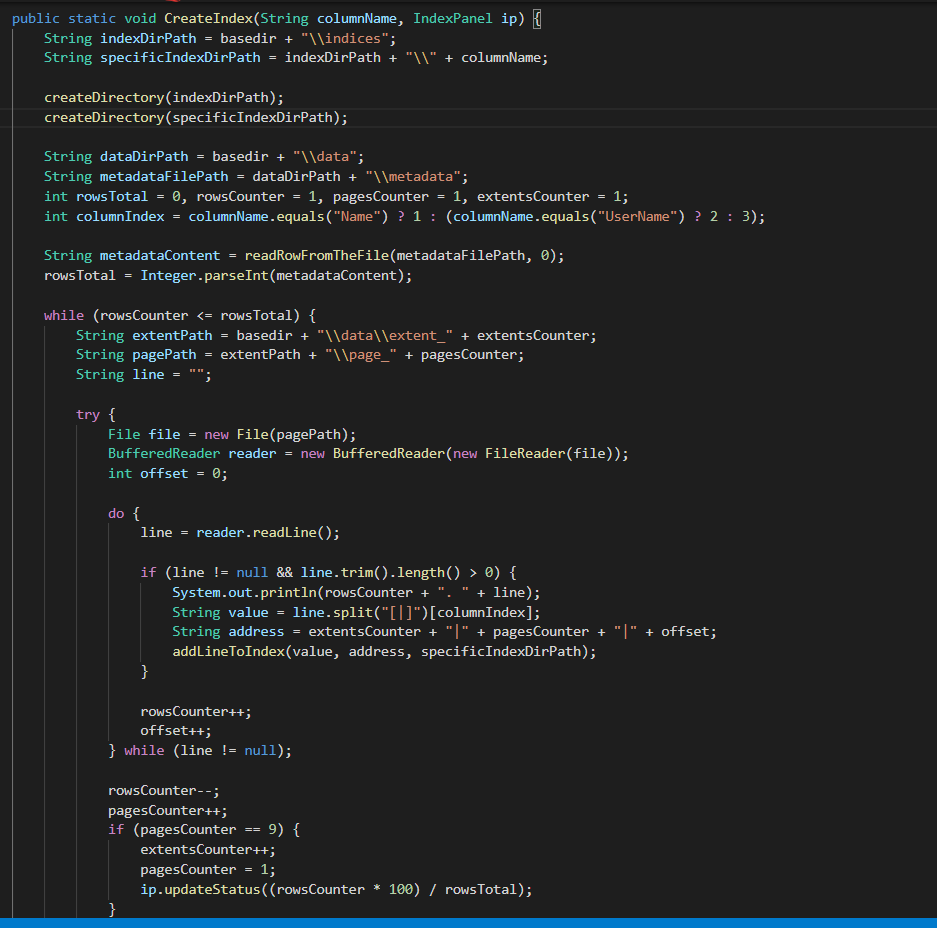


**(fig-4.25: The indices are being created on the Name Field selected)**

****

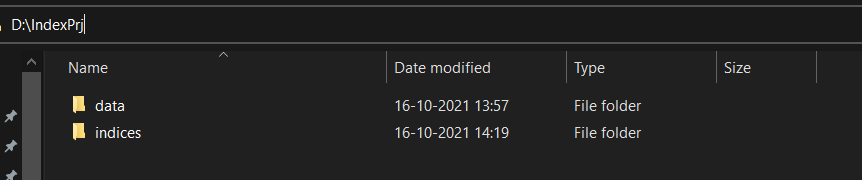
**(fig-4.26: The progress bar updates and message box appears when the indices are created)**

The entire logic of creating the indies and updating of the progress bar is present inside the DataManager class.



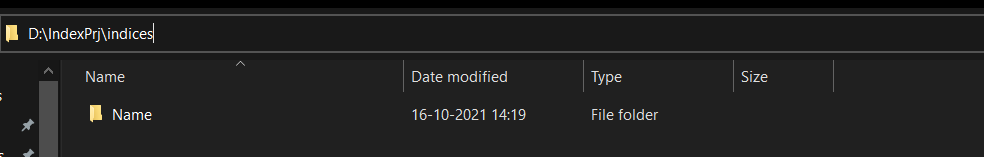
**(fig-4.27: Logic for creating the indices and updating the progress bar)**

Inside the directory IndexPrj that we created, besides the data directory, we now also have the indices directory.



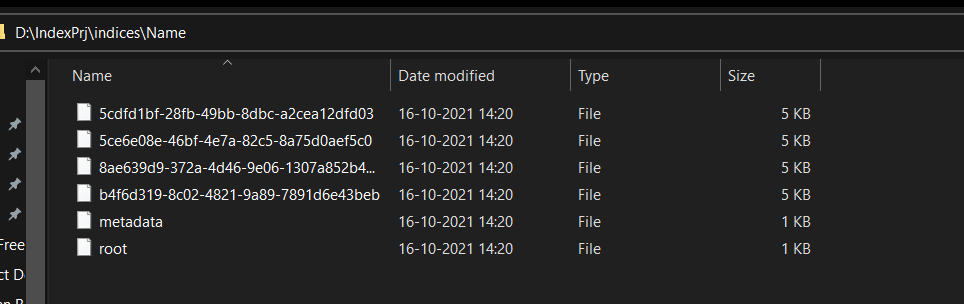
**(fig-4.28: indices directory inside the IndexPrj)**

Inside the indices directory, we have the Name directory because we created the indices upon Name.



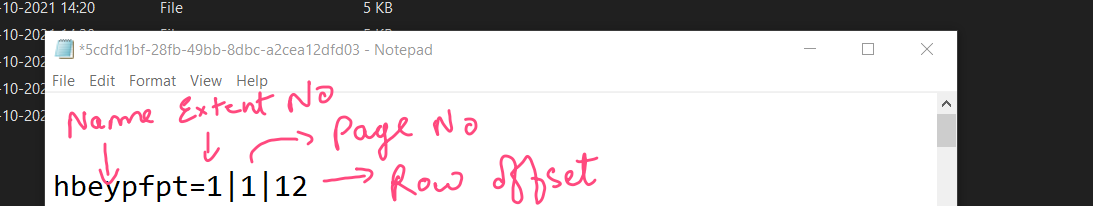
**(fig-4.29: Name directory inside indices)**

Inside the Name folder, we have the index pages, the root index page and the metadata file for the indices created.



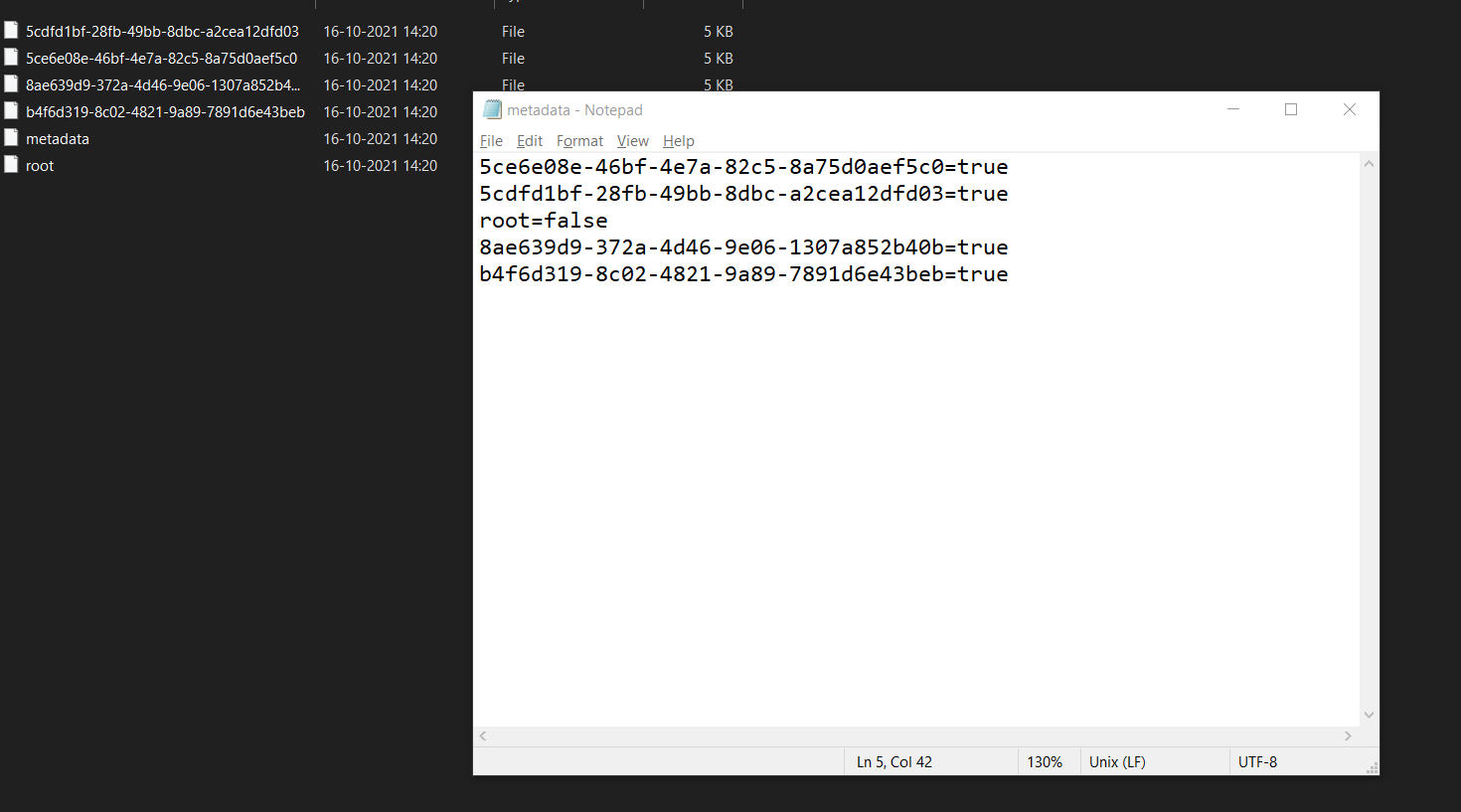
**(fig-4.30: Root, Metadata and the other index pages inside the Name Directory)**

Each data element in the index pages stores the Name (because the indices were created on them) and the extent number, data page number and the row offset in which the data will be found.

****

**(fig-4.31: Index Page data element)**

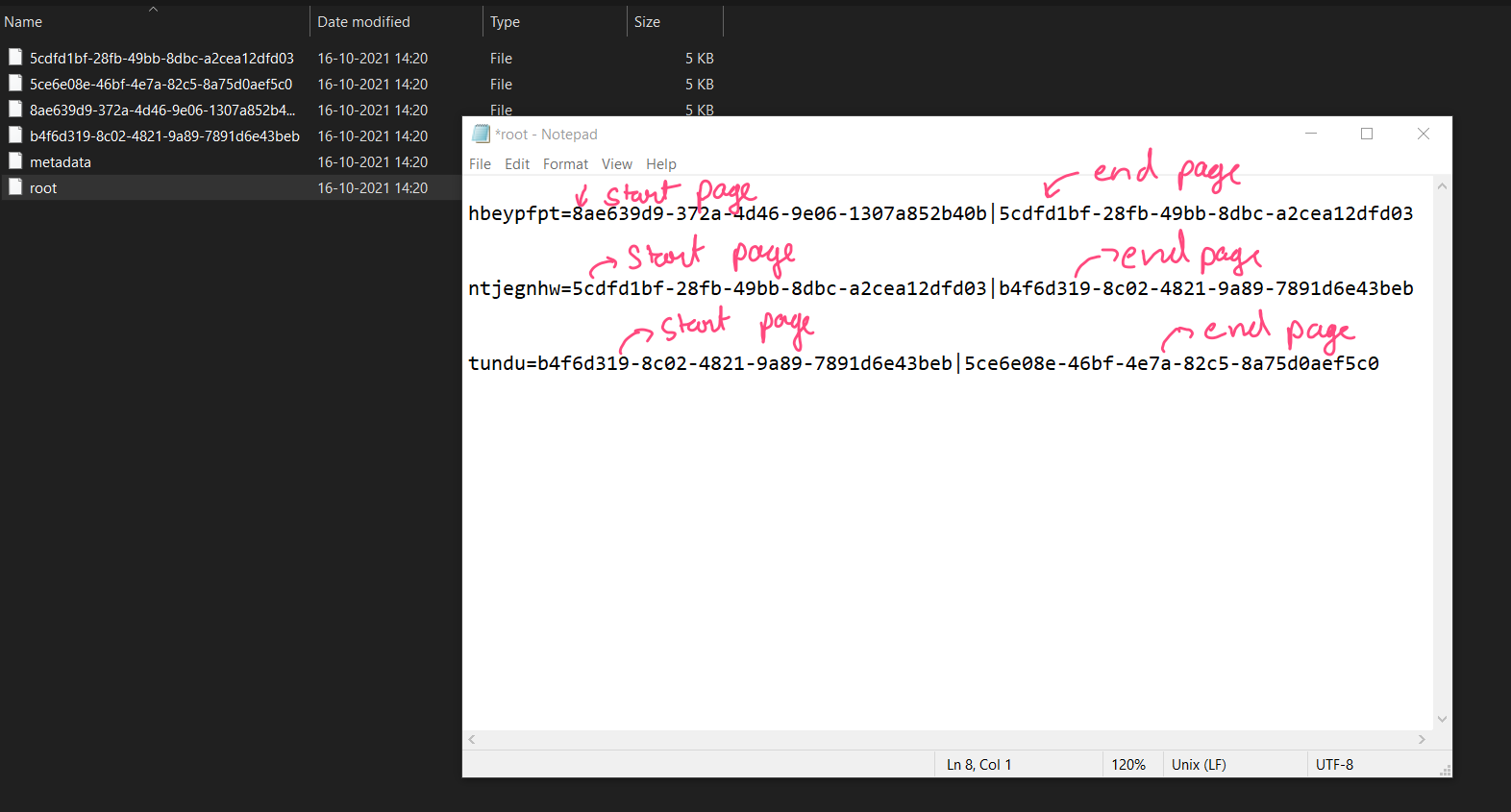
The metadata file here shows the pages that are at the leaf of the B tree as shown below:



**(fig-4.32: Metadata file in the index pages)**

The true value in front of all the data elements depict that they are leaves in the B-Tree and the root = false depicts that the root is not a leaf in this case. The root may become leaf if the data set i.e. the number of rows is very small.

The root index page is the root of the B-Tree as shown below:

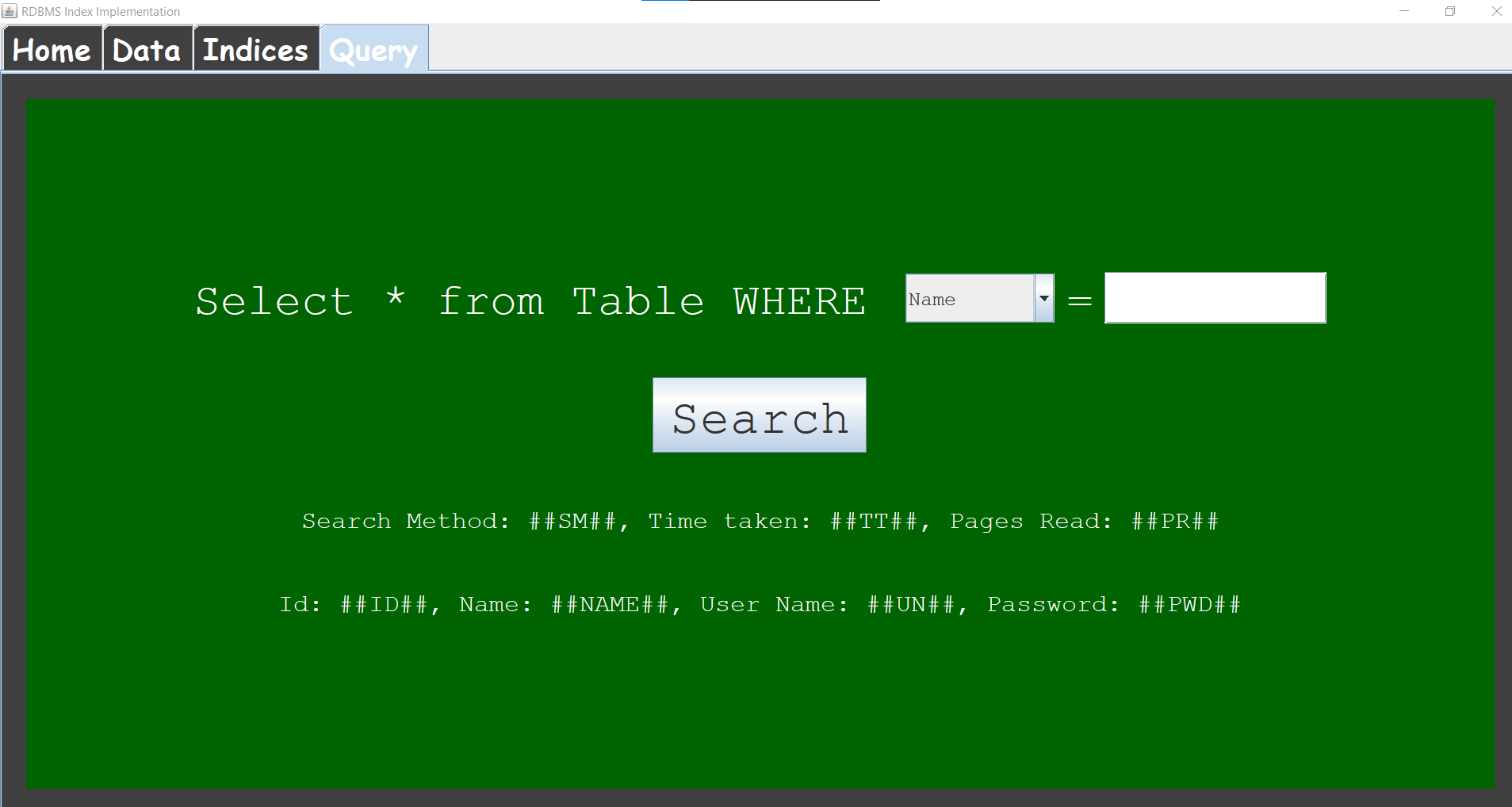


**(fig-4.33: Root Index Page)**

The root index page shows the various divisions inside the root Node of the B-Tree.

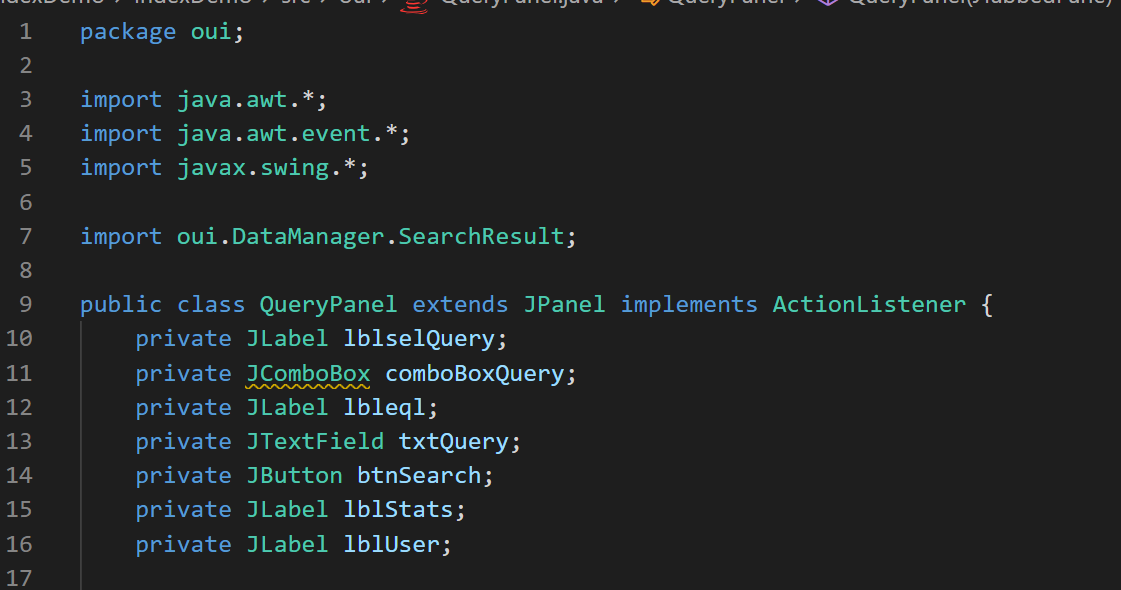
**4.5 Query Panel:**

The Query Panel looks as shown below:



**(fig-4.34: Query Panel)**

Components of the Query Panel are shown below:



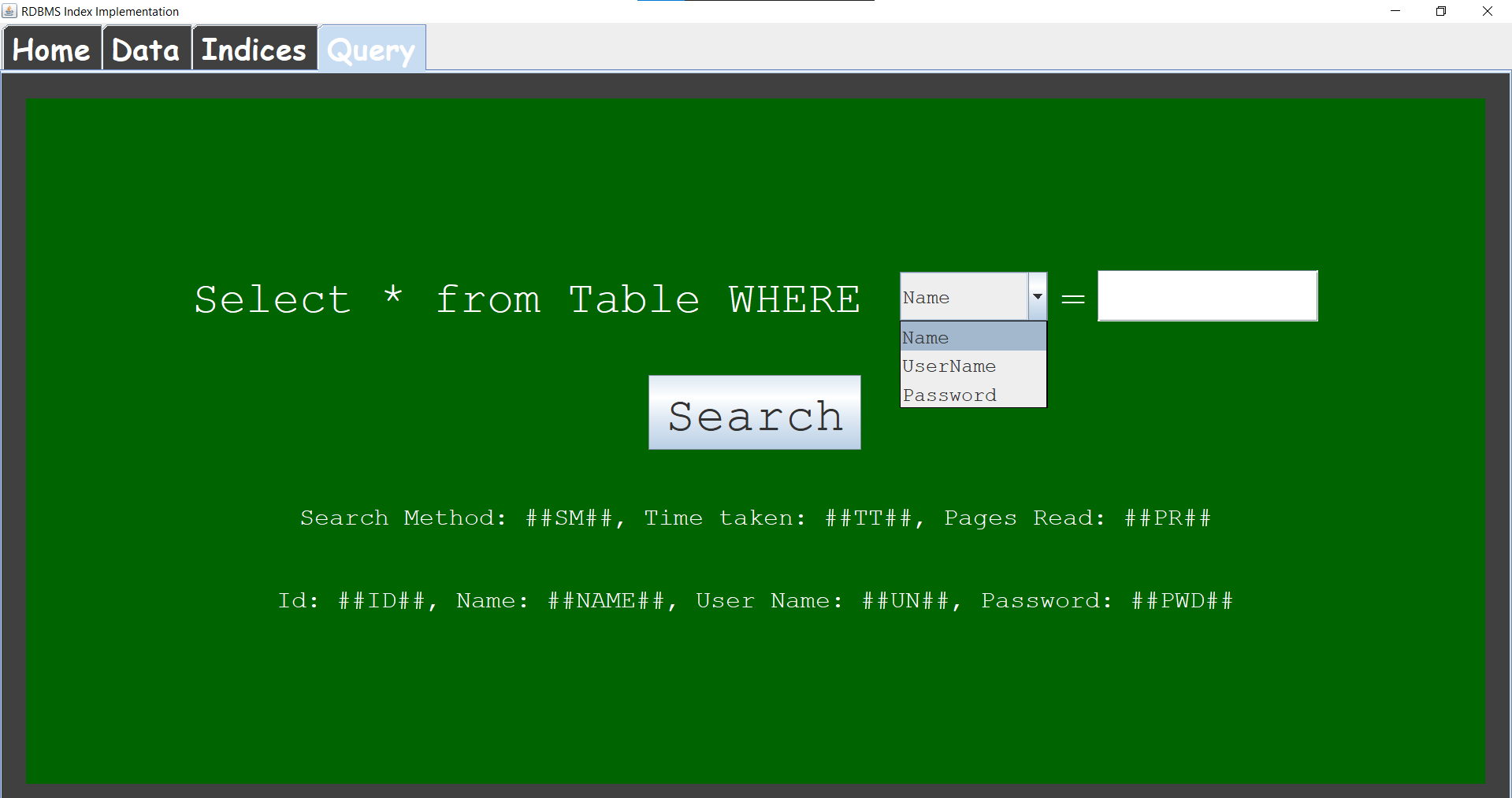
**(fig-4.35: Components of the Query Panel)**

Again, the search query will be performed by clicking on the button and the action of click will be listened by the Query Panel.



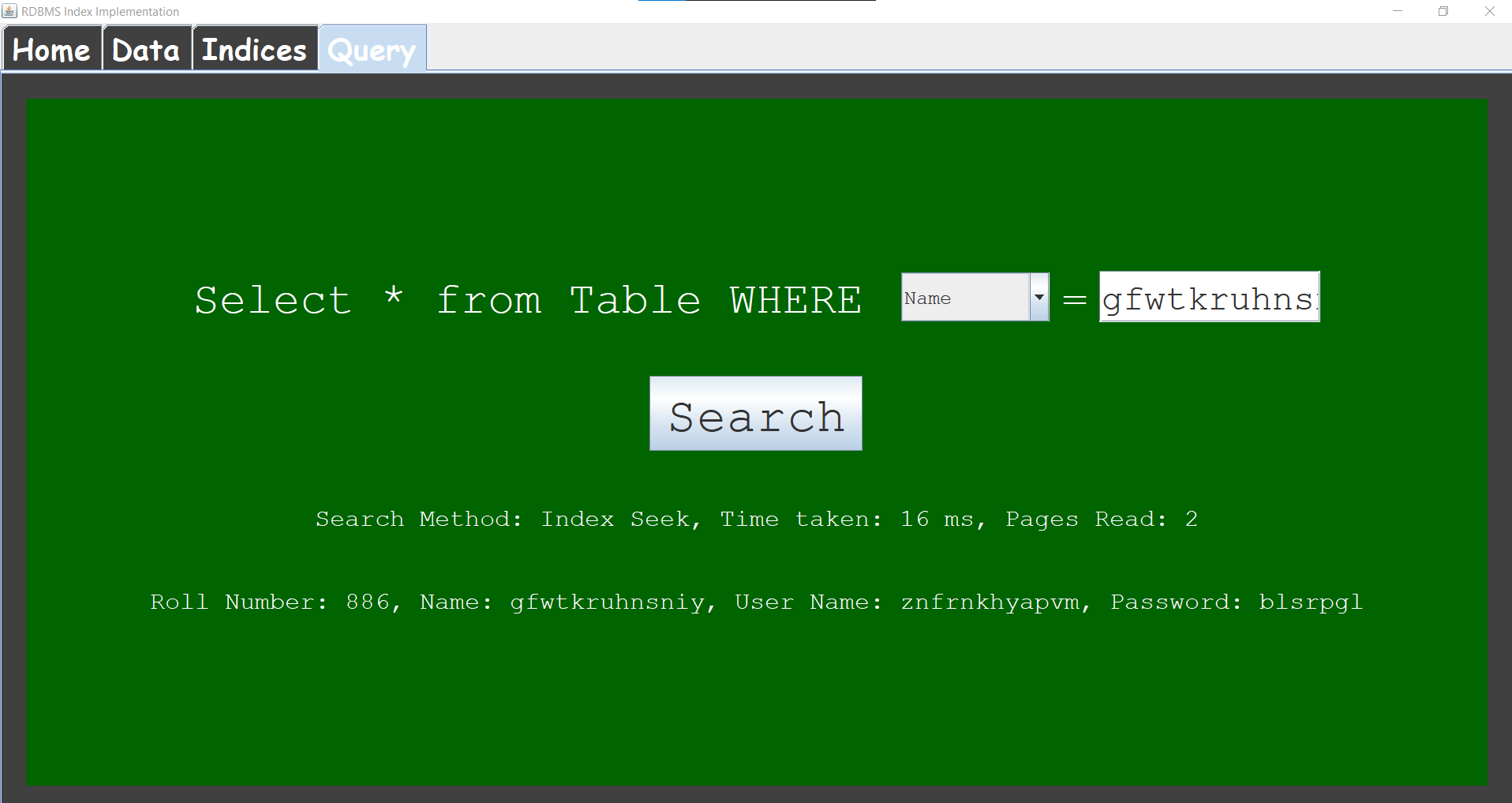
**(fig-4.36: Action Listener attached to the button)**

In the drop down menu, we can select the column in which we want to search. Either the Name, UserName or the password.

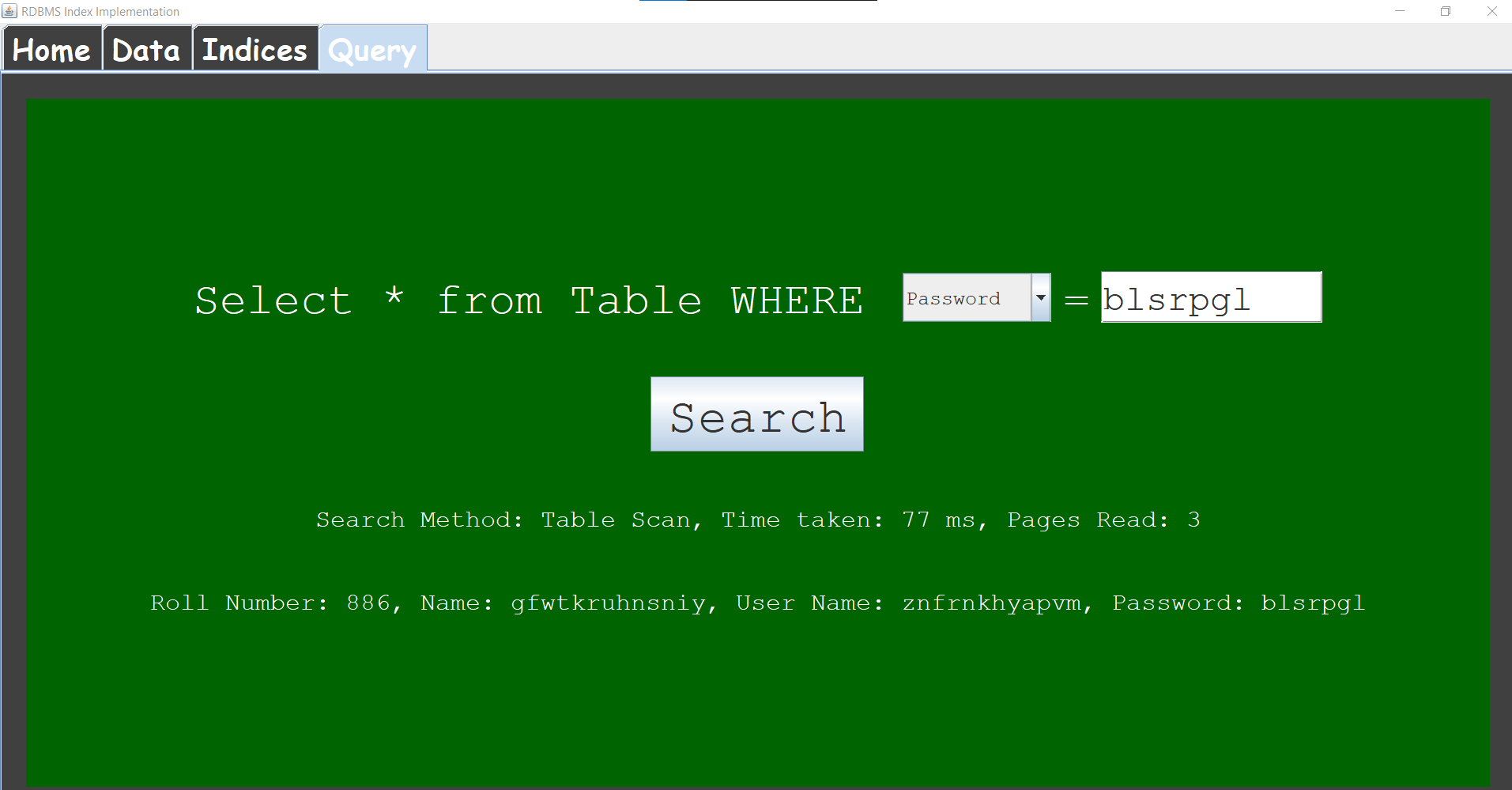


**(fig-4.37: Query Panel dropdown options)**

In the text field, we can enter the data for which we want to search. So, when we select name and search, the search should be faster as we have created indices on name whereas if we search on the basis of password, the time taken will be more and the number of pages searched will also be more as we have not created indices for it. This is shown below:

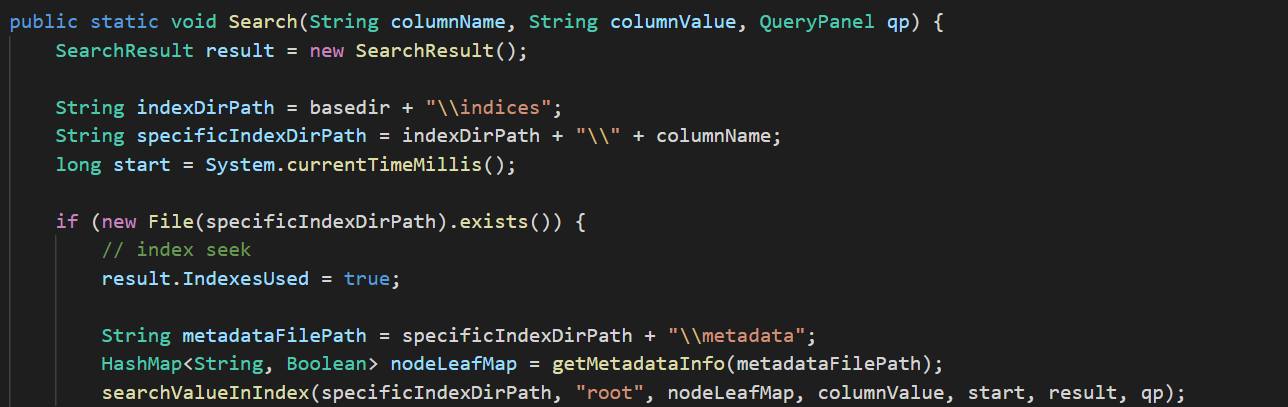


**(fig-4.38: Index Seek method takes 16ms time and searches in two pages)**

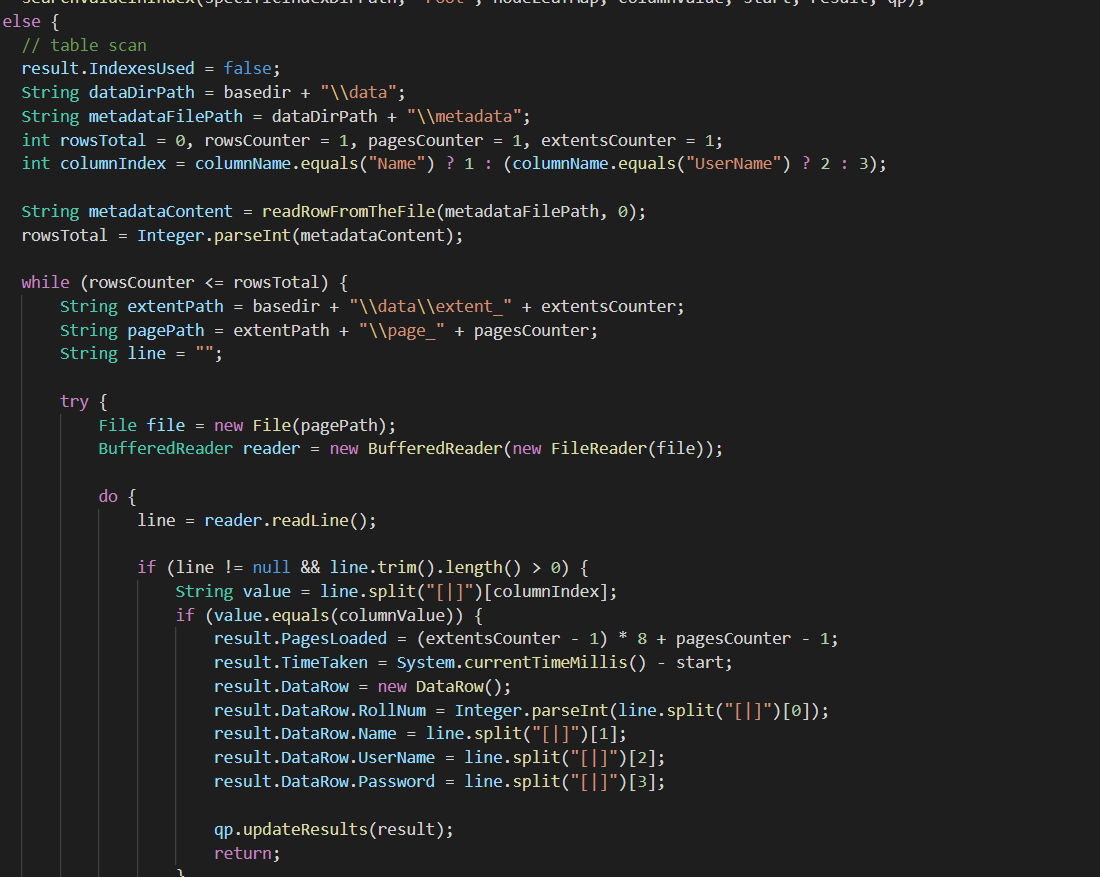


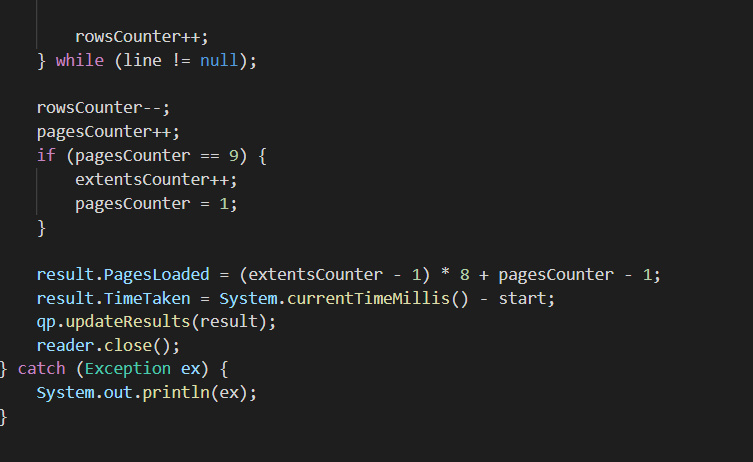
**(fig-4.39: Same data searched by password i.e. Table Scan method reads 3 pages and takes 77ms)**

So, it is clearly seen that the Index Seek method is very fast as compared to the table scan method. The logic for searching is implemented within the DataManager file only.



**(fig-4.40: Index Seek Search Code)**

****

****

**(fig:4.41 (a) and (b): Table Scan Search Code)**

**Chapter-5**

**Conclusion**

It is concluded that disk-access times are much slower than main memory access times. Typical seek times and rotational delays are of the order of 5 to 6 milliseconds and typical data transfer rates are of the range of 5 to 10 million bytes per second and therefore, main memory access times are likely to be at least 4 or 5 orders of magnitude faster than disk access on any given system. Therefore, the objective is to minimize the number of disk accesses and thus, this project is concerned with techniques for achieving that objective i.e. techniques for arranging the data on a disk so that any required piece of data, say some specific record, can be located in a few I/O’s as possible

1. From the above observations, it is very clear that B+tree is better than normal indexing in every possible way.

2. Hence it is always desirable to implement B+ tree data structure to search data in an efficient manner.

3. Multilevel Indexing is Better for larger data whereas sparse indexing does well with smaller data

**Bibliography**

1. Database System Concepts taught in class and text reference textbook by Abraham Silberschatz, Henry F. Korth, and S. Sudarshan

2.Bulk loading in <https://en.wikipedia.org/wiki/B>

3.B+Tree in <https://en.wikipedia.org/wiki/B%2B_tree>

4.Dynamic Programming: <https://en.wikipedia.org/wiki/Dynamic_programming>

5. Donald Kossmann and Konrad Stocker, “Iterative Dynamic Programming: A New Class of Query Optimization Algorithms”.

# 6. Data Structures and Algorithms in Java, Michael T.Goodrich, RobertoTamassia, Michael H. Goldwasser