```
package oui;

import java.awt.*;
import javax.swing.*;

public class App extends JFrame {
	JTabbedPane tabPane;
	HomePanel homePanel;
	DataPanel dataPanel;
	IndexPanel indexPanel;
	QueryPanel queryPanel;

	public App() {
		tabPane = new JTabbedPane();
		tabPane.setFont(new Font("Comic Sans MS", 1, 30));
		tabPane.setBackground(Color.DARK_GRAY);
		tabPane.setForeground(Color.WHITE);

		homePanel = new HomePanel(tabPane);
		dataPanel = new DataPanel(tabPane);
		indexPanel = new IndexPanel(tabPane);
		queryPanel = new QueryPanel(tabPane);

		super.add(tabPane);

		pack();

		super.setTitle("RDBMS Index Implementation");
		super.setExtendedState(MAXIMIZED_BOTH);
		super.setDefaultCloseOperation(DISPOSE_ON_CLOSE);
		super.setVisible(true);
	}
}
	package oui;

	/**
	 *  The {@code BTree} class represents an ordered symbol table of generic
	 *  key-value pairs.
	 *  It supports the <em>put</em>, <em>get</em>, <em>contains</em>,
	 *  <em>size</em>, and <em>is-empty</em> methods.
	 *  A symbol table implements the <em>associative array</em> abstraction:
	 *  when associating a value with a key that is already in the symbol table,
	 *  the convention is to replace the old value with the new value.
	 *  Unlike {@link java.util.Map}, this class uses the convention that
	 *  values cannot be {@code null}—setting the
	 *  value associated with a key to {@code null} is equivalent to deleting the key
	 *  from the symbol table.
	 *  <p>
	 *  This implementation uses a B-tree. It requires that
	 *  the key type implements the {@code Comparable} interface and calls the
	 *  {@code compareTo()} and method to compare two keys. It does not call either
```

```
 *  {@code equals()} or {@code hashCode()}.
 *  The <em>get</em>, <em>put</em>, and <em>contains</em> operations
 *  each make log<sub><em>m</em></sub>(<em>n</em>) probes in the worst case,
 *  where <em>n</em> is the number of key-value pairs
 *  and <em>m</em> is the branching factor.
 *  The <em>size</em>, and <em>is-empty</em> operations take constant time.
 *  Construction takes constant time.
 *  <p>
 *  For additional documentation, see
 *  <a href="https://algs4.cs.princeton.edu/62btree">Section 6.2</a> of
 *  <i>Algorithms, 4th Edition</i> by Robert Sedgewick and Kevin Wayne.
 */
public class BTree<Key extends Comparable<Key>, Value>  {
   // max children per B-tree node = M-1
   // (must be even and greater than 2)
   private static final int M = 4;

   private Node root;       // root of the B-tree
   private int height;      // height of the B-tree
   private int n;           // number of key-value pairs in the B-tree

   // helper B-tree node data type


private static final class Node {

      private int m;                          // number of children
      private Entry[] children = new Entry[M];   // the array of children

      // create a node with k children
      private Node(int k) {

         m = k;
      }
   }

   // internal nodes: only use key and next
   // external nodes: only use key and value

   private static class Entry {

      private Comparable key;
      private final Object val;
      private Node next;     // helper field to iterate over array entries
      public Entry(Comparable key, Object val, Node next) {

         this.key  = key;
         this.val  = val;
         this.next = next;
      }
   }
```

```
        /**

         * Initializes an empty B-tree.
         */

        public BTree() {

           root = new Node(0);
        }



        /**

  * Returns the value associated with the given key.
         *
         * @param  key the key
         * @return the value associated with the given key if the key is in the symbol table
         *         and {@code null} if the key is not in the symbol table
         * @throws IllegalArgumentException if {@code key} is {@code null}
         */


public Value get(Key key) {


           if (key == null) throw new IllegalArgumentException("argument to get() is null");
           return search(root, key, height);
        }




private Value search(Node x, Key key, int ht) {

           Entry[] children = x.children;

           // external node
           if (ht == 0) {
              for (int j = 0; j < x.m; j++) {
                 if (eq(key, children[j].key)) return (Value) children[j].val;
              }
           }

           // internal node
           else {
              for (int j = 0; j < x.m; j++) {
                 if (j+1 == x.m || less(key, children[j+1].key))
                    return search(children[j].next, key, ht-1);
              }
```

```
        }
        return null;
    }


    /**

     * Inserts the key-value pair into the symbol table, overwriting the old value
     * with the new value if the key is already in the symbol table.
     * If the value is {@code null}, this effectively deletes the key from the symbol table.
     *
     * @param  key the key
     * @param  val the value
     * @throws IllegalArgumentException if {@code key} is {@code null}
     */

  public void put(Key key, Value val) {
        if (key == null) throw new IllegalArgumentException("argument key to put() is null");
        Node u = insert(root, key, val, height);
        n++;
        if (u == null) return;

        // need to split root
        Node t = new Node(2);
        t.children[0] = new Entry(root.children[0].key, null, root);
        t.children[1] = new Entry(u.children[0].key, null, u);
        root = t;
        height++;
    }


    private Node insert(Node h, Key key, Value val, int ht) {
        int j;
        Entry t = new Entry(key, val, null);
//traverssal ke liye hai
        // external node
        if (ht == 0) {
            for (j = 0; j < h.m; j++) {
                if (less(key, h.children[j].key)) break;// uski loc mil jae to
            }
        }

        // internal node
        else {
            for (j = 0; j < h.m; j++) {
                if ((j+1 == h.m) || less(key, h.children[j+1].key)) {
                    Node u = insert(h.children[j++].next, key, val, ht-1);
                    if (u == null) return null;/// splitting nahi hui
                    t.key = u.children[0].key;// splitting hui to use iska child bna do
                    t.next = u;
                    break;
```

```
                    }
                  }
                }
/// shifting ke liye
                for (int i = h.m; i > j; i--)
                  h.children[i] = h.children[i-1];
                h.children[j] = t;
                h.m++;
                if (h.m < M) return null;
                else         return split(h);
              }

              // split node in half

private Node split(Node h) {
                Node t = new Node(M/2);
                h.m = M/2;
                for (int j = 0; j < M/2; j++)
                  t.children[j] = h.children[M/2+j];
                return t;
              }


              // comparison functions - make Comparable instead of Key to avoid casts
              private boolean less(Comparable k1, Comparable k2) {
                return k1.compareTo(k2) < 0;
              }

              private boolean eq(Comparable k1, Comparable k2) {
                return k1.compareTo(k2) == 0;
              }


              /**
               * Unit tests the {@code BTree} data type.
               *
               * @param args the command-line arguments
               */

            }
package oui;

import java.io.*;

import java.util.*;

import javax.print.attribute.standard.Media;
import javax.sound.midi.MetaEventListener;

public class DataManager {
          public static String basedir = "";
```

```
	public static class DataRow implements Serializable {

		public int RollNum;
		public String Name;
		public String UserName;
		public String Password;

		public String toString() {

			return this.RollNum + "|" + this.Name + "|" + this.UserName + "|" +
this.Password;
		}
	}


	public static class SearchResult {
		public DataRow DataRow;
		public long TimeTaken;
		public int PagesLoaded;
		public boolean IndexesUsed; // false means scan, true means seek
	}

	public static void CreateData(int numRows, DataPanel dp) throws Exception {
		createDirectory(basedir + "\\data");

		int extentNumber = 1, pageNumber = 1;
		for (int i = 1; i <= numRows; i++) {
			String extentPath = basedir + "\\data\\extent_" + extentNumber;
			String pagePath = basedir + "\\data\\extent_" + extentNumber + "\\page_"
+ pageNumber;

			// checks if the directory and file doesnot exist and creates if not
			createDirectory(extentPath);
			createFile(pagePath);

			// by the time code is here extent directory and file are guaranteed
			// created.
			DataRow row = new DataRow();

			row.RollNum = i;
			row.Name = getRandomWord();
			row.UserName = getRandomWord();
			row.Password = getRandomWord();

			if (canThePageAccomodateARow(pagePath, row)) {
				writeTheRowInTheFile(pagePath, row.toString());
			} else {
				i--;
				pageNumber++;
```

```
                        if (pageNumber == 9) {
                                extentNumber++;
                                pageNumber = 1;

                                dp.updateStatus((i * 100) / numRows);
                        }
                }
        }

        dp.updateStatus(100);
        String metadataFilePath = basedir + "\\data\\metadata";
        createFile(metadataFilePath);
        String metadata = numRows + "";
        writeTheRowInTheFile(metadataFilePath, metadata);
    }



    public static void CreateIndex(String columnName, IndexPanel ip) {
        String indexDirPath = basedir + "\\indices";
        String specificIndexDirPath = indexDirPath + "\\" + columnName;

        createDirectory(indexDirPath);
        createDirectory(specificIndexDirPath);

        String dataDirPath = basedir + "\\data";
        String metadataFilePath = dataDirPath + "\\metadata";
        int rowsTotal = 0, rowsCounter = 1, pagesCounter = 1, extentsCounter = 1;
        int columnIndex = columnName.equals("Name") ? 1 :
(columnName.equals("UserName") ? 2 : 3);

        String metadataContent = readRowFromTheFile(metadataFilePath, 0);
        rowsTotal = Integer.parseInt(metadataContent);

        while (rowsCounter <= rowsTotal) {
            String extentPath = basedir + "\\data\\extent_" + extentsCounter;
            String pagePath = extentPath + "\\page_" + pagesCounter;
            String line = "";

            try {
                File file = new File(pagePath);
                BufferedReader reader = new BufferedReader(new FileReader(file));
                int offset = 0;

                do {
                    line = reader.readLine();

                    if (line != null && line.trim().length() > 0) {
                        System.out.println(rowsCounter + ". " + line);
                        String value = line.split("[|]")[columnIndex];
```

```
                                        String address = extentsCounter + "|" +
pagesCounter + "|" + offset;
                                        addLineToIndex(value, address,
specificIndexDirPath);
                                }

                                rowsCounter++;
                                offset++;
                        } while (line != null);

                        rowsCounter--;
                        pagesCounter++;
                        if (pagesCounter == 9) {
                                extentsCounter++;
                                pagesCounter = 1;
                                ip.updateStatus((rowsCounter * 100) / rowsTotal);
                        }

                        reader.close();
                } catch (Exception ex) {
                        System.out.println(ex);
                }
        }

        ip.updateStatus(100);
    }


    public static void Search(String columnName, String columnValue, QueryPanel qp) {
        SearchResult result = new SearchResult();

        String indexDirPath = basedir + "\\indices";
        String specificIndexDirPath = indexDirPath + "\\" + columnName;
        long start = System.currentTimeMillis();

        if(new File(specificIndexDirPath).exists()){
            // index seek
            result.IndexesUsed = true;

            String metadataFilePath = specificIndexDirPath + "\\metadata";
            HashMap<String, Boolean> nodeLeafMap =
getMetadataInfo(metadataFilePath);
            searchValueInIndex(specificIndexDirPath, "root", nodeLeafMap,
columnValue, start, result, qp);
        } else {
            // table scan
            result.IndexesUsed = false;
            String dataDirPath = basedir + "\\data";
            String metadataFilePath = dataDirPath + "\\metadata";
            int rowsTotal = 0, rowsCounter = 1, pagesCounter = 1, extentsCounter = 1;
```

```
                        int columnIndex = columnName.equals("Name") ? 1 :
(columnName.equals("UserName") ? 2 : 3);

                        String metadataContent = readRowFromTheFile(metadataFilePath, 0);
                        rowsTotal = Integer.parseInt(metadataContent);

                        while (rowsCounter <= rowsTotal) {
                                String extentPath = basedir + "\\data\\extent_" + extentsCounter;
                                String pagePath = extentPath + "\\page_" + pagesCounter;
                                String line = "";

                                try {
                                        File file = new File(pagePath);
                                        BufferedReader reader = new BufferedReader(new
FileReader(file));

                                        do {
                                                line = reader.readLine();

                                                if (line != null && line.trim().length() > 0) {
                                                        String value = line.split("[|]")[columnIndex];
                                                        if(value.equals(columnValue)){
                                                                result.PagesLoaded =
(extentsCounter - 1) * 8 + pagesCounter - 1;
                                                                result.TimeTaken =
System.currentTimeMillis() - start;
                                                                result.DataRow = new DataRow();
                                                                result.DataRow.RollNum =
Integer.parseInt(line.split("[|]")[0]);
                                                                result.DataRow.Name =
line.split("[|]")[1];
                                                                result.DataRow.UserName =
line.split("[|]")[2];
                                                                result.DataRow.Password =
line.split("[|]")[3];

                                                                qp.updateResults(result);
                                                                return;
                                                        }
                                                }

                                                rowsCounter++;
                                        } while (line != null);

                                        rowsCounter--;
                                        pagesCounter++;
                                        if (pagesCounter == 9) {
                                                extentsCounter++;
                                                pagesCounter = 1;
                                        }
```

```
                                        result.PagesLoaded = (extentsCounter - 1) * 8 +
pagesCounter - 1;
                                        result.TimeTaken = System.currentTimeMillis() - start;
                                        qp.updateResults(result);
                                        reader.close();
                                } catch (Exception ex) {
                                        System.out.println(ex);
                                }
                        }

                }
        }


        private static String readRowFromTheFile(String fileName, int rowNumber) {
                String line = "";

                try {
                        File file = new File(fileName);
                        BufferedReader reader = new BufferedReader(new FileReader(file));

                        int rowCounter = 1;
                        do {
                                line = reader.readLine();
                                rowCounter++;
                        } while (line != null && rowCounter <= rowNumber);

                        reader.close();
                } catch (Exception ex) {
                        System.out.println(ex);
                }

                return line;
        }

        private static void writeTheRowInTheFile(String fileName, String row) {
                try {
                        File file = new File(fileName);
                        BufferedWriter writer = new BufferedWriter(new FileWriter(file, true));
                        PrintWriter pwriter = new PrintWriter(writer);
                        pwriter.println(row);
                        pwriter.close();
                } catch (Exception ex) {
                        System.out.println(ex);
                }
        }

        private static boolean canThePageAccomodateARow(String fileName, DataRow row) {
                File file = new File(fileName);
                long byteSize = file.length() + row.toString().length();
                return byteSize < 8 * 1024;
```

```
}

private static boolean doesFileExist(String fileName) {
	File file = new File(fileName);
	return file.exists();
}

private static void createFile(String fileName) {
	if (!doesFileExist(fileName)) {
		try {
			File file = new File(fileName);
			file.createNewFile();
		} catch (IOException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
	}
}

private static boolean doesDirectoryExist(String dirName) {
	File file = new File(dirName);
	return file.exists();
}

private static void createDirectory(String dirName) {
	if (!doesDirectoryExist(dirName)) {
		File dir = new File(dirName);
		dir.mkdir();
	}
}


private static String getRandomWord() {
	int length = (int) (Math.random() * 10) + 5;
	StringBuilder sb = new StringBuilder();

	for (int i = 0; i < length; i++) {
		char ch = (char) ((int) (Math.random() * 26) + 'a');
		sb.append(ch);
	}

	return sb.toString();
}

private static void addLineToIndex(String value, String address, String indexDirPath) {
	String metadataFilePath = indexDirPath + "\\metadata";
	createFile(metadataFilePath);

	String rootFilePath = indexDirPath + "\\root";
	createRootFile(rootFilePath, metadataFilePath);
```

```
		// guarantee that the root and metadata file exists file exists
		HashMap<String, Boolean> nodeLeafMap = getMetadataInfo(metadataFilePath);
		addLineToNode(indexDirPath, "root", nodeLeafMap, value, address);
		manageChildNodeSize(indexDirPath, "root", null, nodeLeafMap);
		updateMetadataInfo(metadataFilePath, nodeLeafMap);
	}

	private static void addLineToNode(String indexDirPath, String node, HashMap<String,
Boolean> nodeLeafMap,
			String value, String address) {
		Boolean isLeaf = nodeLeafMap.get(node);

		if (!isLeaf) {
			// search place and go down
			String childNode = "";

			try {
				File file = new File(indexDirPath + "\\" + node);
				BufferedReader reader = new BufferedReader(new FileReader(file));
				String prevLine = "", line = "", lv = "";
				boolean flag = false;

				do {
					prevLine = line;
					line = reader.readLine();

					if (line != null && line.trim().length() > 0) {
						lv = line.split("=")[0];

						if (lv.compareTo(value) > 0) {
							flag = true;
							break;
						}
					}
				} while (line != null);

				if (flag) {
					// smaller
					childNode = line.split("=")[1].split("[|]")[0];
				} else {
					// larger
					childNode = prevLine.split("=")[1].split("[|]")[1];
				}

				reader.close();
			} catch (Exception ex) {
				System.out.println(ex);
			}

			addLineToNode(indexDirPath, childNode, nodeLeafMap, value, address);
			manageChildNodeSize(indexDirPath, childNode, node, nodeLeafMap);
```

```
	} else {
		// add
		try {
			File file = new File(indexDirPath + "\\" + node);
			BufferedReader reader = new BufferedReader(new FileReader(file));
			ArrayList<String> lines = new ArrayList<>();
			String line = "", lv = "";
			boolean flag = false;

			do {
				line = reader.readLine();

				if (line != null && line.trim().length() > 0) {
					lv = line.split("=")[0];

					if (lv.compareTo(value) > 0) {
						if (flag == false) {
							lines.add(value + "=" + address);
							lines.add(line);
							flag = true;
						} else {
							lines.add(line);
						}
					} else {
						lines.add(line);
					}
				}
			} while (line != null);

			reader.close();

			if (flag == false) {
				lines.add(value + "=" + address);
			}

			BufferedWriter writer = new BufferedWriter(new FileWriter(file));
			for (String lineContent : lines) {
				writer.write(lineContent);
				writer.write("\n");
			}
			writer.close();
		} catch (Exception ex) {
			System.out.println(ex);
		}
	}
}

private static void manageChildNodeSize(String indexDirPath, String childNode, String node,
		HashMap<String, Boolean> nodeLeafMap) {
	File file = new File(indexDirPath + "\\" + childNode);
	long fileSize = file.length();
```

```
		if (fileSize > 8 * 1024) {
			// split
			String siblingNode = UUID.randomUUID().toString();
			createFile(indexDirPath + "\\" + siblingNode);
			ArrayList<String> lines = new ArrayList<>();

			try {
				// reading data from child
				BufferedReader reader = new BufferedReader(new FileReader(file));
				String line = "";

				do {
					line = reader.readLine();
					if (line != null && line.trim().length() > 0) {
						lines.add(line);
					}
				} while (line != null);

				reader.close();

				// half the data in original child
				BufferedWriter writer = new BufferedWriter(new FileWriter(file));
				for (int i = 0; i < lines.size() / 2; i++) {
					writer.write(lines.get(i));
					writer.write("\n");
				}
				writer.close();

				if (childNode == "root") {
					childNode = UUID.randomUUID().toString();
					file.renameTo(new File(indexDirPath + "\\" + childNode));

					nodeLeafMap.put(childNode, nodeLeafMap.get("root"));
					nodeLeafMap.put(siblingNode, nodeLeafMap.get("root"));
					nodeLeafMap.put("root", false);
				} else {
					nodeLeafMap.put(siblingNode,
nodeLeafMap.get(childNode));
				}

				// half the data in new child
				writer = new BufferedWriter(new FileWriter(new File(indexDirPath +
"\\" + siblingNode)));
				for (int i = lines.size() / 2; i < lines.size(); i++) {
					writer.write(lines.get(i));
					writer.write("\n");
				}
				writer.close();

				// reading and preparing content from node
				String lineTI = lines.get(lines.size() / 2);
```

```
                                        lines.clear();

                                        if (node != null) {
                                                boolean flag = false;
                                                reader = new BufferedReader(new FileReader(new
File(indexDirPath + "\\" + node)));
                                                do {
                                                        line = reader.readLine();
                                                        if (line != null && line.trim().length() > 0) {
                                                                if (line.compareTo(lineTI) < 0) {
                                                                        lines.add(line);
                                                                } else {
                                                                        if (flag == false) {
                                                                                flag = true;
                                                                                lines.add(lineTI.split("=")[0]
+ "=" + childNode + "|" + siblingNode);
                                                                                lines.add(

        line.split("=")[0] + "=" + siblingNode + "|" + line.split("=")[1].split("[|]")[1]);
                                                                        } else {
                                                                                lines.add(line);
                                                                        }
                                                                }
                                                        }
                                                } while (line != null);

                                                if (flag == false) {
                                                        lines.add(lineTI.split("=")[0] + "=" + childNode + "|"
+ siblingNode);
                                                }
                                                reader.close();

                                                // insert content in node
                                                writer = new BufferedWriter(new FileWriter(new
File(indexDirPath + "\\" + node)));
                                                for (int i = 0; i < lines.size(); i++) {
                                                        writer.write(lines.get(i));
                                                        writer.write("\n");
                                                }
                                                writer.close();
                                        } else {
                                                createFile(indexDirPath + "\\root");
                                                writer = new BufferedWriter(new FileWriter(new
File(indexDirPath + "\\root")));
                                                writer.write(lineTI.split("=")[0] + "=" + childNode + "|" +
siblingNode);
                                                writer.write("\n");
                                                writer.close();
                                        }
                                } catch (Exception ex) {
                                        System.out.println(ex);
```

```
                }
            }
        }


private static void searchValueInIndex(String indexDirPath, String node,

        HashMap<String, Boolean> nodeLeafMap, String columnValue,
                                                                                long start,
SearchResult result, QueryPanel qp) {
            Boolean isLeaf = nodeLeafMap.get(node);

            if (!isLeaf) {
                String childNode = "";

                try {
                    File file = new File(indexDirPath + "\\" + node);
                    BufferedReader reader = new BufferedReader(new FileReader(file));
                    String prevLine = "", line = "", lv = "";
                    boolean flag = false;

                    do {
                        prevLine = line;
                        line = reader.readLine();

                        if (line != null && line.trim().length() > 0) {
                            lv = line.split("=")[0];

                            if (lv.compareTo(columnValue) > 0) {
                                flag = true;
                                break;
                            }
                        }
                    } while (line != null);

                    if (flag) {
                        // smaller
                        childNode = line.split("=")[1].split("[|]")[0];
                    } else {
                        // larger
                        childNode = prevLine.split("=")[1].split("[|]")[1];
                    }

                    reader.close();
                } catch (Exception ex) {
                    System.out.println(ex);
                }


                result.PagesLoaded++;
```

```
                        result.TimeTaken = System.currentTimeMillis() - start;
                        qp.updateResults(result);
                        searchValueInIndex(indexDirPath, childNode, nodeLeafMap, columnValue,
start, result, qp);
                } else {
                        String childNode = "";

                        try {
                                File file = new File(indexDirPath + "\\" + node);
                                BufferedReader reader = new BufferedReader(new FileReader(file));
                                String prevLine = "", line = "", lv = "";

                                do {
                                        prevLine = line;
                                        line = reader.readLine();

                                        if (line != null && line.trim().length() > 0) {
                                                lv = line.split("=")[0];

                                                if (lv.compareTo(columnValue) == 0) {
                                                        break;
                                                }
                                        }
                                } while (line != null);

                                reader.close();

                                String extentNumber = line.split("=")[1].split("[|]")[0];
                                String pageNumber = line.split("=")[1].split("[|]")[1];
                                int offsetNumber = Integer.parseInt(line.split("=")[1].split("[|]")[2]);

                                File dataFilePath = new File(basedir + "\\data\\extent_" +
extentNumber + "\\page_" + pageNumber);
                                reader = new BufferedReader(new FileReader(dataFilePath));

                                for(int i = 0; i <= offsetNumber; i++){
                                        line = reader.readLine();
                                }

                                result.PagesLoaded++;
                                result.TimeTaken = System.currentTimeMillis() - start;
                                result.DataRow = new DataRow();
                                result.DataRow.RollNum = Integer.parseInt(line.split("[|]")[0]);
                                result.DataRow.Name = line.split("[|]")[1];
                                result.DataRow.UserName = line.split("[|]")[2];
                                result.DataRow.Password = line.split("[|]")[3];
                                qp.updateResults(result);
                                reader.close();
                        } catch (Exception ex) {
                                System.out.println(ex);
                        }
```

```
		}
	}

	private static HashMap<String, Boolean> getMetadataInfo(String metadataFilePath) {
		HashMap<String, Boolean> map = new HashMap<>();

		try {
			File file = new File(metadataFilePath);
			BufferedReader reader = new BufferedReader(new FileReader(file));

			String line = "";
			do {
				line = reader.readLine();
				if (line != null) {
					map.put(line.split("=")[0],
Boolean.parseBoolean(line.split("=")[1]));
				}
			} while (line != null);
			reader.close();
		} catch (Exception ex) {

		}

		return map;
	}

	private static void updateMetadataInfo(String metadataFilePath, HashMap<String, Boolean>
nodeLeafMap) {
		try {
			BufferedWriter writer = new BufferedWriter(new FileWriter(new
File(metadataFilePath)));
			Set<String> keys = nodeLeafMap.keySet();
			for (String key : keys) {
				writer.write(key + "=" + nodeLeafMap.get(key));
				writer.write("\n");
			}
			writer.close();
		} catch (Exception ex) {
		}
	}

	private static void createRootFile(String rootFilePath, String metadataFilePath) {
		if (!doesFileExist(rootFilePath)) {
			try {
				File file = new File(metadataFilePath);
				BufferedWriter writer = new BufferedWriter(new FileWriter(file,
true));
				PrintWriter pwriter = new PrintWriter(writer);
				pwriter.println("root=true"); // root is leaf to begin with
				pwriter.close();
```

```
                                file = new File(rootFilePath);
                                file.createNewFile();
                        } catch (IOException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }
                }
        }
}
package oui;

import java.awt.*;
import javax.swing.*;
import javax.swing.border.Border;

import java.awt.event.*;

public class DataPanel extends JPanel implements ActionListener {
        private JLabel lblRows;
        private JTextField txtRows;
        private JButton btnCreate;
        private JProgressBar progressBar;

        public DataPanel(JTabbedPane tabPane) {
                tabPane.addTab("Data", this);
                initComponents();
        }

        private void initComponents() {
                this.setBorder(BorderFactory.createMatteBorder(25, 25, 25, 25, Color.darkGray));
                this.setBackground(new Color(0, 100, 0));

                lblRows = new JLabel();
                lblRows.setFont(new Font("Monospaced", 0, 50)); // NOI18N
                lblRows.setForeground(new Color(255, 255, 255));
                lblRows.setText("Rows:");

                txtRows = new JTextField("");
                txtRows.setColumns(10);
                txtRows.setFont(new Font("Monospaced", 0, 50)); // NOI18N

                btnCreate = new JButton();
                btnCreate.setFont(new Font("Monospaced", 0, 50)); // NOI18N
                btnCreate.setText("Create Test Data");
                btnCreate.addActionListener(this);

                progressBar = new JProgressBar();
                progressBar.setStringPainted(true);
                progressBar.setMinimum(0);
                progressBar.setMaximum(100);
                progressBar.setFont(new Font("Monospaced", 0, 50));
```

```
        GroupLayout layout = new GroupLayout(this);
        this.setLayout(layout);

    layout.setHorizontalGroup(layout.createParallelGroup(GroupLayout.Alignment.LEADING)

    .addGroup(layout.createSequentialGroup().addContainerGap(GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                                        .addComponent(lblRows,
GroupLayout.PREFERRED_SIZE, GroupLayout.DEFAULT_SIZE,
                                                GroupLayout.PREFERRED_SIZE)

    .addPreferredGap(LayoutStyle.ComponentPlacement.RELATED)
                                        .addComponent(txtRows,
GroupLayout.PREFERRED_SIZE, GroupLayout.DEFAULT_SIZE,
                                                GroupLayout.PREFERRED_SIZE)
                                        .addContainerGap(GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))

    .addGroup(layout.createSequentialGroup().addContainerGap(GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                                        .addComponent(btnCreate,
GroupLayout.PREFERRED_SIZE, GroupLayout.DEFAULT_SIZE,
                                                GroupLayout.PREFERRED_SIZE)
                                        .addContainerGap(GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))

    .addGroup(layout.createSequentialGroup().addContainerGap(GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                                        .addComponent(progressBar,
GroupLayout.PREFERRED_SIZE, GroupLayout.DEFAULT_SIZE,
                                                GroupLayout.PREFERRED_SIZE)
                                        .addContainerGap(GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)));
        layout.setVerticalGroup(

    layout.createSequentialGroup().addContainerGap(GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)

    .addGroup(layout.createParallelGroup(GroupLayout.Alignment.BASELINE)
                                                .addComponent(txtRows,
GroupLayout.PREFERRED_SIZE, GroupLayout.DEFAULT_SIZE,

    GroupLayout.PREFERRED_SIZE)
                                                .addComponent(lblRows,
GroupLayout.PREFERRED_SIZE, GroupLayout.DEFAULT_SIZE,

    GroupLayout.PREFERRED_SIZE))
                            .addGap(50, 50, 50)
                            .addComponent(btnCreate, GroupLayout.PREFERRED_SIZE,
GroupLayout.DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE)
```

```
                                .addGap(50, 50, 50)
                                .addComponent(progressBar, GroupLayout.PREFERRED_SIZE,
GroupLayout.DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE)
                                .addContainerGap(GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE));

        }

        @Override
        public void actionPerformed(ActionEvent e) {
                DataPanel obj = this;
                Thread t = new Thread(new Runnable() {
                        @Override
                        public void run() {
                                try {
                                        int rows = Integer.parseInt(txtRows.getText());
                                        DataManager.CreateData(rows, obj);
                                        JOptionPane.showMessageDialog(obj, "Data created
successfully");
                                        progressBar.setValue(0);
                                } catch (Exception ex) {
                                }
                        }
                });
                t.start();
        }

        public void updateStatus(int percent){
                progressBar.setValue(percent);
        }
}
package oui;

import java.awt.*;
import javax.swing.*;

public class HomePanel extends javax.swing.JPanel {
   public HomePanel(JTabbedPane tabPane) {
        tabPane.addTab("Home", this);
      initComponents();
   }

   private JTextArea jTextArea;
   private void initComponents() {
        this.setBorder(BorderFactory.createMatteBorder(25, 25, 25, 25, Color.darkGray));
        this.setBackground(new Color(0, 100, 0));

      jTextArea = new JTextArea();
      jTextArea.setFont(new Font("Monospaced", 0, 35)); // NOI18N
      jTextArea.setForeground(new Color(102, 0, 102));
      jTextArea.setColumns(60);
```

```
      jTextArea.setRows(5);
      jTextArea.setText("In this project,I am making a software which will make searching in
Relational Database Management system optimized. As well as we will also show better
performance of Index seek method over Table scan method.");
      jTextArea.setLineWrap(true);
      jTextArea.setEditable(false);

      javax.swing.GroupLayout layout = new GroupLayout(this);
      this.setLayout(layout);
      layout.setHorizontalGroup(
         layout.createParallelGroup(GroupLayout.Alignment.LEADING)
         .addGroup(
           layout.createSequentialGroup()
            .addContainerGap(0, Short.MAX_VALUE)
            .addComponent(jTextArea, GroupLayout.PREFERRED_SIZE, GroupLayout.DEFAULT_SIZE,
GroupLayout.PREFERRED_SIZE)
            .addContainerGap(0, Short.MAX_VALUE))
      );
      layout.setVerticalGroup(
         layout.createParallelGroup(GroupLayout.Alignment.LEADING)
         .addGroup(
           layout.createSequentialGroup()
            .addContainerGap(0, Short.MAX_VALUE)
            .addComponent(jTextArea, GroupLayout.PREFERRED_SIZE, GroupLayout.DEFAULT_SIZE,
GroupLayout.PREFERRED_SIZE)
            .addContainerGap(0, Short.MAX_VALUE))
      );
   }
}
package oui;

import java.awt.*;
import java.awt.event.*;

import javax.swing.*;

public class IndexPanel extends JPanel implements ActionListener {
        private JLabel lblRows;
        private JComboBox comboIndices;
        private JButton btnCreate;
        private JProgressBar progressBar;

        public IndexPanel(JTabbedPane tabPane) {
                tabPane.addTab("Indices", this);
                initComponents();
        }

        private void initComponents() {
                this.setBorder(BorderFactory.createMatteBorder(25, 25, 25, 25, Color.darkGray));
                this.setBackground(new Color(0, 100, 0));
```

```
                lblRows = new JLabel();
                lblRows.setFont(new Font("Monospaced", 0, 50)); // NOI18N
                lblRows.setForeground(new Color(255, 255, 255));
                lblRows.setText("Column:");

                comboIndices = new JComboBox(new String[] {"--Select One--", "Name",
"UserName", "Password"});
                comboIndices.setFont(new Font("Monospaced", 0, 35)); // NOI18N

                btnCreate = new JButton();
                btnCreate.setFont(new Font("Monospaced", 0, 50)); // NOI18N
                btnCreate.setText("Create Index");
                btnCreate.addActionListener(this);

                progressBar = new JProgressBar();
                progressBar.setStringPainted(true);
                progressBar.setMinimum(0);
                progressBar.setMaximum(100);
                progressBar.setFont(new Font("Monospaced", 0, 50));

                GroupLayout layout = new GroupLayout(this);
                this.setLayout(layout);
                layout.setHorizontalGroup(
                        layout.createParallelGroup(GroupLayout.Alignment.LEADING)
                        .addGroup(
                                        layout.createSequentialGroup()
                                        .addContainerGap(GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                                        .addComponent(lblRows, GroupLayout.PREFERRED_SIZE,
GroupLayout.DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE)

                .addPreferredGap(LayoutStyle.ComponentPlacement.RELATED)
                                        .addComponent(comboIndices,
GroupLayout.PREFERRED_SIZE, GroupLayout.DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE)
                                        .addContainerGap(GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
                        .addGroup(
                                        layout.createSequentialGroup()
                                        .addContainerGap(GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                                        .addComponent(btnCreate, GroupLayout.PREFERRED_SIZE,
GroupLayout.DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE)
                                        .addContainerGap(GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))

                .addGroup(layout.createSequentialGroup().addContainerGap(GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                                         .addComponent(progressBar,
GroupLayout.PREFERRED_SIZE, GroupLayout.DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE)
                                         .addContainerGap(GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)));
```

```
		layout.setVerticalGroup(
			layout.createSequentialGroup()
			.addContainerGap(GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
			.addGroup(

	layout.createParallelGroup(GroupLayout.Alignment.BASELINE)
					.addComponent(comboIndices,
GroupLayout.PREFERRED_SIZE, GroupLayout.DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE)
					.addComponent(lblRows, GroupLayout.PREFERRED_SIZE,
GroupLayout.DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE))
			.addGap(50, 50, 50)
			.addComponent(btnCreate, GroupLayout.PREFERRED_SIZE,
GroupLayout.DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE)
			.addGap(50, 50, 50)
			.addComponent(progressBar, GroupLayout.PREFERRED_SIZE,
GroupLayout.DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE)
			.addContainerGap(GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE));

	}

	@Override
	public void actionPerformed(ActionEvent e) {
		IndexPanel obj = this;
		Thread t = new Thread(new Runnable() {
			@Override
			public void run() {
				try {
					if(comboIndices.getSelectedIndex() == 0){
						JOptionPane.showMessageDialog(obj, "Please select
a column");
						return;
					}

	DataManager.CreateIndex(comboIndices.getSelectedItem().toString(), obj);
					JOptionPane.showMessageDialog(obj, "Index created
successfully");
					progressBar.setValue(0);
				} catch (Exception ex) {
				}
			}
		});
		t.start();
	}

	public void updateStatus(int percent){
		progressBar.setValue(percent);
	}
}
package oui;

public class mybtree {
```

```
	private static final int M = 4;

	private static final class Node {
		private int m;
		private int[] children = new int[m];

		private Node(int m) {
			this.m = m;
		}
	}

	private Node root; // root of the B-tree
	private int height; // height of the B-tree
	private int n;

}
package oui;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

import oui.DataManager.SearchResult;

public class QueryPanel extends JPanel implements ActionListener {
	private JLabel lblselQuery;
	private JComboBox comboBoxQuery;
	private JLabel lbleql;
	private JTextField txtQuery;
	private JButton btnSearch;
	private JLabel lblStats;
	private JLabel lblUser;

	public QueryPanel(JTabbedPane tabPane) {
		tabPane.addTab("Query", this);
		initComponents();
	}

	private void initComponents() {
		this.setBorder(BorderFactory.createMatteBorder(25, 25, 25, 25, Color.darkGray));
		this.setBackground(new Color(0, 100, 0));

		lblselQuery = new JLabel();
		lblselQuery.setFont(new Font("Monospaced", 0, 45));
		lblselQuery.setForeground(new Color(255, 255, 255));
		lblselQuery.setText("Select * from Table WHERE ");

		String[] option = { "Name", "UserName", "Password" };
		comboBoxQuery = new JComboBox<>(option);
		comboBoxQuery.setFont(new Font("Monospaced", 0, 20));
		comboBoxQuery.setPreferredSize(new Dimension(150, 50));
```

```
                lbleql = new JLabel();
                lbleql.setFont(new Font("Monospaced", 0, 45));
                lbleql.setForeground(new Color(255, 255, 255));
                lbleql.setText("=");

                txtQuery = new JTextField("");
                txtQuery.setColumns(10);
                txtQuery.setFont(new Font("Monospaced", 0, 36));

                btnSearch = new JButton();
                btnSearch.setFont(new Font("Monospaced", 0, 50));
                btnSearch.setText("Search");
                btnSearch.addActionListener(this);

                lblStats = new JLabel();
                lblStats.setFont(new Font("Monospaced", 0, 25));
                lblStats.setForeground(new Color(255, 255, 255));
                lblStats.setText("Search Method: ##SM##, Time taken: ##TT##, Pages Read:
##PR##");

                lblUser = new JLabel();
                lblUser.setFont(new Font("Monospaced", 0, 25));
                lblUser.setForeground(new Color(255, 255, 255));
                lblUser.setText("Id: ##ID##, Name: ##NAME##, User Name: ##UN##, Password:
##PWD##");

                GroupLayout layout = new GroupLayout(this);
                this.setLayout(layout);

        layout.setHorizontalGroup(layout.createParallelGroup(GroupLayout.Alignment.LEADING)

        .addGroup(layout.createSequentialGroup().addContainerGap(GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                                                .addComponent(lblselQuery,
GroupLayout.PREFERRED_SIZE, GroupLayout.DEFAULT_SIZE,
                                                        GroupLayout.PREFERRED_SIZE)
                                .addPreferredGap(LayoutStyle.ComponentPlacement.RELATED)
                                .addComponent(comboBoxQuery, GroupLayout.PREFERRED_SIZE,
GroupLayout.DEFAULT_SIZE,

        GroupLayout.PREFERRED_SIZE).addPreferredGap(LayoutStyle.ComponentPlacement.RELATE
D)
                                .addComponent(lbleql, GroupLayout.PREFERRED_SIZE,
GroupLayout.DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE)

        .addPreferredGap(LayoutStyle.ComponentPlacement.RELATED).addComponent(txtQuery,
                                                GroupLayout.PREFERRED_SIZE,
GroupLayout.DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE)
                                .addContainerGap(GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
```

```
                .addGroup(layout.createSequentialGroup()
                        .addContainerGap(GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE).addComponent(btnSearch,
                                GroupLayout.PREFERRED_SIZE,
GroupLayout.DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE)
                        .addContainerGap(GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))

                .addGroup(layout.createSequentialGroup()
                        .addContainerGap(GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE).addComponent(lblStats,
                                GroupLayout.PREFERRED_SIZE,
GroupLayout.DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE)
                        .addContainerGap(GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))

                .addGroup(layout.createSequentialGroup()
                        .addContainerGap(GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE).addComponent(lblUser,
                                GroupLayout.PREFERRED_SIZE,
GroupLayout.DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE)
                        .addContainerGap(GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
                        );

                layout.setVerticalGroup(
                        layout.createSequentialGroup()
                        .addContainerGap(GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                        .addGroup(

        layout.createParallelGroup(GroupLayout.Alignment.CENTER)
                                .addComponent(lblselQuery,
GroupLayout.PREFERRED_SIZE, GroupLayout.DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE)
                                .addComponent(comboBoxQuery,
GroupLayout.PREFERRED_SIZE, GroupLayout.DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE)
                                .addComponent(lbleql,
GroupLayout.PREFERRED_SIZE, GroupLayout.DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE)
                                .addComponent(txtQuery,
GroupLayout.PREFERRED_SIZE, GroupLayout.DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE))
                        .addGap(50, 50, 50)
                        .addComponent(btnSearch, GroupLayout.PREFERRED_SIZE,
GroupLayout.DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE)
                        .addGap(50, 50, 50)
                        .addComponent(lblStats, GroupLayout.PREFERRED_SIZE,
GroupLayout.DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE)
                        .addGap(50, 50, 50)
                        .addComponent(lblUser, GroupLayout.PREFERRED_SIZE,
GroupLayout.DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE)
                        .addContainerGap(GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE));
        }

        @Override
        public void actionPerformed(ActionEvent e) {
```

```
                DataManager.Search(comboBoxQuery.getSelectedItem().toString(),
txtQuery.getText(), this);
        }

        public void updateResults(SearchResult result)
        {
                if(result.DataRow != null){
                        lblUser.setText
                        (
                                        "Roll Number: ##RNO##, Name: ##NAME##, User Name:
##UN##, Password: ##PWD##"
                                        .replace("##RNO##", result.DataRow.RollNum + "\n")
                                        .replace("##NAME##", result.DataRow.Name + "\n")
                                        .replace("##UN##", result.DataRow.UserName + "\n")
                                        .replace("##PWD##", result.DataRow.Password + "\n")
                        );
                }
                lblStats.setText
                (
                                "Search Method: ##SM##, Time taken: ##TT##, Pages Read:
##PR##"
                                .replace("##SM##", result.IndexesUsed? "Index Seek\n": "Table
Scan\n")
                                .replace("##TT##", result.TimeTaken + " ms\n")
                                .replace("##PR##", result.PagesLoaded + "\n")
                );
        }
}
import oui.*;

public class Client {

        public static void main(String[] args) throws Exception {
                DataManager.basedir = args[0];
                App app = new App();
        }
}
```