**User Manual**

**For**

**Database Management System**

**Database Management System: A Comprehensive Tool for Tabular Data Management**

**Submitted by**

**Muhammad Rizwan Shafiq, SP24-BCS-069**

**Yousaf Allahdad, SP24-BCS-097**

**Department of Computer Science, CU, Islamabad.**

**7th January,2025**

Table of Contents

[Abstract iii](#_Toc186938412)

[1. Introduction 1](#_Toc186938413)

[2. Scope of the Project 1](#_Toc186938414)

[3. Getting Started 1](#_Toc186938415)

[Installation 1](#_Toc186938416)

[Running the Program 1](#_Toc186938417)

[4. Commands Overview 2](#_Toc186938418)

[CREATE 2](#_Toc186938419)

[DROP 2](#_Toc186938420)

[SHOW 2](#_Toc186938421)

[INSERT 2](#_Toc186938422)

[SELECT 2](#_Toc186938423)

[UPDATE 3](#_Toc186938424)

[DELETE 3](#_Toc186938425)

[HELP 3](#_Toc186938426)

[EXIT 3](#_Toc186938427)

[5. Example Usage 4](#_Toc186938428)

[6. System Limitations 4](#_Toc186938429)

[7. Tools and Technologies 4](#_Toc186938430)

[8. Error Handling 5](#_Toc186938431)

[9. Java Code (Appendix) 5](#_Toc186938432)

**Project Category: (**Desktop Application/Information System**)**

# Abstract

This Database Management System (DBMS) is a command-line application designed to provide an efficient and user-friendly way to manage tabular data. It enables users to create, delete, and modify tables stored as .txt files, providing a lightweight and accessible alternative to more complex database management systems. By supporting a variety of custom query commands, such as creating tables, inserting data, updating records, and retrieving specific information with optional filtering and sorting, the system offers a practical tool for managing structured data. The project is tailored for educational purposes and small-scale projects, showcasing core database management concepts like modular programming, file handling, and basic query processing. Additionally, the system ensures ease of use through its intuitive command syntax and informative error handling, making it suitable for beginners seeking to explore database management systems.

# Introduction

The Database Management System (DBMS) simplifies the handling of structured data by offering a minimalistic query language. Users can create tables with specific columns, insert data, update existing records, and retrieve or delete data based on conditions. This manual is designed to guide users through the functionalities of DBMS, explain its commands, and provide examples for practical understanding.

The project showcases foundational database concepts implemented using Java, highlighting modular programming and file handling. It serves as a steppingstone for those interested in understanding database systems at a fundamental level.

# Scope of the Project

|  |  |
| --- | --- |
| Feature | Supported |
| Create customizable tables | ✅ |
| Insert, update, delete table rows | ✅ |
| Retrieve and filter data | ✅ |
| Advanced database features (joins) | ❌ |
| Authentication and user control | ❌ |

This DBMS is ideal for academic use and small projects requiring basic database functionalities

# Getting Started

## Installation

1. Ensure you have Java (JDK 8 or later) installed on your machine.
2. Download the provided DatabaseManagementSystem.java file.

## Running the Program

1. Compile the program:

javac OurProject.java

1. Run the program:

java OurProject

1. The program will display the following prompt:

>>

Enter commands as described in the "Commands Overview" section.

# Commands Overview

## CREATE

Use this command to create a new table with specified columns.

* **Syntax**: CREATE table\_name HAVING column1, column2, column3,...
* **Example**:

CREATE student HAVING name, regno, address

This command creates a table named student with columns name, regno, and address.

## DROP

Use this command to delete an existing table.

* **Syntax**: DROP table\_name TABLE
* **Example**:

DROP student TABLE

Deletes the student table after confirmation.

## SHOW

Use this command to display all available tables.

* **Syntax**: SHOW TABLES
* **Example**: SHOW TABLES

## INSERT

Use this command to insert a new row into a table.

* **Syntax**: INSERT INTO table\_name VALUES ("value1", "value2", ...)
* **Example**:

INSERT INTO student VALUES ("Ali", "CS20-123", "Lahore")

Adds a new row to the student table.

## SELECT

Use this command to retrieve rows from a table, with optional filtering and sorting.

* **Syntax**: SELECT FROM table\_name WHERE column=value ORDER BY column
* **Example**:

SELECT FROM student WHERE address="Lahore" ORDER BY name

Retrieves rows where address is Lahore and sorts them by name.

**Note**: The WHERE and ORDER BY clauses are optional. If WHERE is omitted, all rows are displayed. If ORDER BY is omitted, rows appear in the order they are stored in the file.

## UPDATE

Use this command to modify specific rows in a table.

* **Syntax**: UPDATE TABLE table\_name SET column TO value HAVING column = value
* **Example**:

UPDATE TABLE student SET address TO "Karachi" HAVING name="Ali"

Updates the address column to Karachi for rows where name is Ali.

## DELETE

Use this command to delete specific rows from a table.

* **Syntax**: DELETE FROM table\_name WHERE column=value
* **Example**:

DELETE FROM TABLE student HAVING name="Ali"

Deletes rows where name is Ali.

## HELP

Displays the syntax for all commands.

* **Command**: HELP

## EXIT

Terminates the program.

* **Command**: EXIT

# Example Usage

>> CREATE student HAVING name, regno, address

Table created successfully: student

>> INSERT INTO student VALUES ("Ali", "CS20-123", "Lahore")

Row inserted successfully into table: student

>> SELECT FROM student WHERE address="Lahore" ORDER BY name

name regno address

-------------------------------------------

Ali CS20-123 Lahore

>> UPDATE TABLE student SET address TO "Karachi" HAVING name="Ali"

1 rows updated successfully in table: student

>> DELETE FROM TABLE student HAVING name="Ali"

Deleted 1 rows from the table: student

>> EXIT

Exiting the program. Goodbye!

# System Limitations

|  |  |
| --- | --- |
| Limitation | Details |
| Single-table operations only | Joins and advanced queries are not supported |
| Limited to plain text file storage | May slow down with large datasets |
| No authentication or user access control | Accessible to all users |

# Tools and Technologies

|  |  |  |
| --- | --- | --- |
| Tools | Version | Purpose |
| Java Development Kit | 8 or later | Program compilation and execution |
| Text Editor | Any | Editing .java files |
| Command Line | N/A | Running the application |

# Error Handling

|  |  |
| --- | --- |
| Error Type | Example Message |
| Invalid Syntax | Error: Invalid DELETE syntax. Use: DELETE FROM TABLE table\_name HAVING column=value |
| Table Not Found | Error: Table not found: student |
| Column Mismatch | Error: Number of values does not match the number of columns in the table. |
| Empty Table | Error: Table is empty or corrupted. |
| Invalid Column Name | Error: Column not found in SET clause: age |
| Duplicate Rows | Error: Duplicate row insertion is not allowed. |
| File Replacement Failure | Error: Failed to rename temporary file to: student.txt |
| Invalid Table Name | Error: Invalid table name: '123Invalid'. It must follow Java identifier rules. |
| Deletion Canceled | Error: Deletion canceled. Table student was not deleted. |

# Java Code (Appendix)

import java.io.\*;

import java.util.Scanner;

public class DatabaseManagementSystem {

    //main and executeCommand

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.println("Welcome to the Database Management System!");

        boolean loopCondition = true;

        while (loopCondition) {

            System.out.print("\n>> ");

            String command = sc.nextLine();

            loopCondition = executeCommand(command, sc);

        }

        sc.close();

    }

    public static boolean executeCommand(String command, Scanner sc) {

        try {

            String[] partsOfString = command.split(" ", 2);

            String actionToExecute = partsOfString[0].toUpperCase();

            switch (actionToExecute) {

                case "CREATE":

                    handleCreateCase(partsOfString[1]);

                    break;

                case "DROP":

                    handleDropCase(partsOfString[1], sc);

                    break;

                case "SHOW":

                    handleShowCase(partsOfString[1]);

                    break;

                case "INSERT":

                    handleInsertCase(partsOfString[1]);

                    break;

                case "SELECT":

                    handleSelectCase(partsOfString[1]);

                    break;

                case "UPDATE":

                    handleUpdateCase(partsOfString[1]);

                    break;

                case "DELETE":

                    handleDeleteCase(partsOfString[1]);

                    break;

                case "HELP":

                    displayHelp();

                    break;

                case "EXIT":

                    System.out.println("Exiting the program. Goodbye!");

                    return false; // It will terminate the loop

                default:

                    throw new IllegalArgumentException("Unknown command. Try Again!");

            }

        } catch (IOException | IllegalArgumentException e) {

            System.out.println("Error: " + e.getMessage());

        }

        return true; // It will continue the loop by default

    }

    //Methods for All Statements

    public static void handleCreateCase(String remainingString) throws IOException {

        String lowerCaseString = remainingString.toLowerCase();

        if (!lowerCaseString.contains(" having ")) {

            throw new IllegalArgumentException("Invalid CREATE syntax. Use: CREATE table\_name HAVING column1, column2, ...");

        }

        int havingIndex = lowerCaseString.indexOf(" having ");

        String tableName = remainingString.substring(0, havingIndex).trim();

        String columns = remainingString.substring(havingIndex + 7).trim();

        // Validate table name

        if (!satisfiesJavaIdentifierRules(tableName)) {

            throw new IllegalArgumentException("Invalid table name: '" + tableName + "'. It must follow Java identifier rules and cannot be a reserved Java keyword.");

        }

        if (columns.endsWith(",")) {

            throw new IllegalArgumentException("Column names must not end with a comma.");

        }

        String[] columnArray = columns.split(",");

        for (int i = 0; i < columnArray.length; i++) {

            columnArray[i] = columnArray[i].trim();

            if (columnArray[i].isEmpty()) {

                throw new IllegalArgumentException("Column names cannot be empty.");

            }

            // Validate column names

            if (!satisfiesJavaIdentifierRules(columnArray[i])) {

                throw new IllegalArgumentException("Invalid column name: '" + columnArray[i] + "'. It must follow Java identifier rules and cannot be a reserved Java keyword.");

            }

        }

        File directory = new File("DBMS\_TABLES");

        if (!directory.exists()) {

            directory.mkdir();

        }

        File file = new File("DBMS\_TABLES/" + tableName + ".txt");

        if (file.exists()) {

            throw new IOException("Table already exists: " + tableName);

        }

        String header = String.join(",", columnArray);

        try (PrintWriter writer = new PrintWriter(file)) {

            writer.write(header + "\n");

            System.out.println("Table created successfully: " + tableName);

        }

    }

    public static void handleDropCase(String remainingString, Scanner sc) throws FileNotFoundException {

        String lowerCaseString = remainingString.toLowerCase();

        if (!lowerCaseString.endsWith(" table")) {

            throw new IllegalArgumentException("Invalid DROP syntax. Use: DROP table\_name TABLE");

        }

        String tableName = remainingString.split(" ")[0].trim();

        // Validate table name

        if (!satisfiesJavaIdentifierRules(tableName)) {

            throw new IllegalArgumentException("Invalid table name: '" + tableName + "'. It must follow Java identifier rules and cannot be a reserved Java keyword.");

        }

        File file = new File("DBMS\_TABLES\\" +tableName+ ".txt");

        System.out.println("Please confirm to delete the Table "+tableName+ "Enter y/n: ");

        String choice = sc.nextLine().trim().toLowerCase();

        if(choice.equals("y")){

            if (file.exists() && file.delete()){

                System.out.println("Successfully Deleted the Table: " +tableName);

            } else {

                throw new FileNotFoundException("Table " +tableName+ " not found!");

            }

        } else {

            System.out.println("Deletion canceled. Table " + tableName + " was not deleted.");

        }

    }

    public static void handleShowCase(String remainingString) {

        if (!remainingString.equalsIgnoreCase("TABLES")) {

            throw new IllegalArgumentException("Invalid SHOW syntax. Use: SHOW TABLES");

        }

        System.out.println("\nAvailable tables:");

        listAllTables("DBMS\_TABLES", "  ");

    }

    public static void handleInsertCase(String remainingString) throws IOException {

        String lowerCaseString = remainingString.toLowerCase();

        if (!lowerCaseString.startsWith("into ") || !lowerCaseString.contains(" values ")) {

            throw new IllegalArgumentException("Invalid INSERT syntax. Use: INSERT INTO table\_name VALUES (\"value1\", \"value2\", ...)");

        }

        int indexAfterInto = lowerCaseString.indexOf("into ") + 5;

        int valuesIndex = lowerCaseString.indexOf(" values ");

        String tableName = remainingString.substring(indexAfterInto, valuesIndex).trim();

        String valuesPart = remainingString.substring(valuesIndex + 8).trim();

        // Validate table name

        if (!satisfiesJavaIdentifierRules(tableName)) {

            throw new IllegalArgumentException("Invalid table name: '" + tableName + "'. It must follow Java identifier rules and cannot be a reserved Java keyword.");

        }

        if (!valuesPart.startsWith("(") || !valuesPart.endsWith(")")) {

            throw new IllegalArgumentException("Values must be enclosed in parentheses. Example: VALUES (\"value1\", \"value2\", ...)");

        }

        // Extracted values and removed parentheses here

        valuesPart = valuesPart.substring(1, valuesPart.length() - 1).trim();

        String[] values = valuesPart.split(",(?=(?:[^\"]\*\"[^\"]\*\")\*[^\"]\*$)");

        for (int i = 0; i < values.length; i++) {

            values[i] = values[i].replace("\"", "").trim(); // Remove quotes and trim spaces

        }

        File file = new File("DBMS\_TABLES/" + tableName + ".txt");

        if (!file.exists()) {

            throw new FileNotFoundException("Table not found: " + tableName);

        }

        try (BufferedReader reader = new BufferedReader(new FileReader(file));

            PrintWriter writer = new PrintWriter(new FileWriter(file, true))) {

            String header = reader.readLine();

            if (header == null) {

                throw new IOException("Table is empty or corrupted.");

            }

            String[] columns = header.split(",");

            if (columns.length != values.length) {

                throw new IllegalArgumentException("Number of values does not match the number of columns in the table.");

            }

            // Construct the row to append

            StringBuilder newRow = new StringBuilder();

            for (int i = 0; i < values.length; i++) {

                newRow.append(values[i]);

                if (i < values.length - 1) {

                    newRow.append(",");

                }

            }

            String newRowLower = newRow.toString().toLowerCase().trim();

            String line;

            boolean isDuplicate = false;

            // Check for duplicates

            while ((line = reader.readLine()) != null) {

                if (line.trim().toLowerCase().equals(newRowLower)) {

                    isDuplicate = true;

                    break;

                }

            }

            if (isDuplicate) {

                System.out.println("Error: Duplicate row insertion is not allowed.");

            } else {

                writer.println(newRow.toString());

                System.out.println("Row inserted successfully into table: " + tableName);

            }

        }

    }

    public static void handleSelectCase(String remainingString) throws IOException {

        remainingString = remainingString.toUpperCase();

        // Parse the SELECT command

        String[] splitByOrder = remainingString.split(" ORDER BY ");

        String beforeOrderBy = splitByOrder[0].trim();

        String orderByColm = (splitByOrder.length > 1) ? splitByOrder[1].trim() : null;

        String[] splitWhereClause = beforeOrderBy.split(" WHERE ");

        String tableName = splitWhereClause[0].split("FROM ")[1].trim(); // Extract table name safely

        String whereCondition = (splitWhereClause.length > 1) ? splitWhereClause[1].trim() : null;

        // Validate table name

        if (!satisfiesJavaIdentifierRules(tableName)) {

            throw new IllegalArgumentException("Invalid table name: '" + tableName + "'. It must follow Java identifier rules and cannot be a reserved Java keyword.");

        }

        if (tableName.isEmpty()) {

            throw new IllegalArgumentException("Table name is missing in SELECT command.");

        }

        File file = new File("DBMS\_TABLES\\" + tableName + ".txt");

        if (!file.exists()) {

            throw new FileNotFoundException("Table does not exist: " + tableName);

        }

        try (BufferedReader reader = new BufferedReader(new FileReader(file))) {

            String headerLine = reader.readLine();

            if (headerLine == null) {

                throw new IOException("Table is empty: " + tableName);

            }

            String[] headerColumns = headerLine.split(",");

            int whereColIndex = -1, orderByIndex = -1;

            // Handle WHERE clause

            String whereValue = null;

            if (whereCondition != null) {

                String[] conditionSplit = whereCondition.split("=");

                if (conditionSplit.length != 2) {

                    throw new IllegalArgumentException("Invalid WHERE clause syntax. Use: column = \"value\"");

                }

                String whereColumn = conditionSplit[0].trim();

                whereValue = conditionSplit[1].replace("\"", "").trim(); // Change 2: Extract and normalize whereValue

                whereColIndex = findColumnIndex(headerColumns, whereColumn);

                if (whereColIndex == -1) {

                    throw new IllegalArgumentException("Column not found in WHERE clause: " + whereColumn);

                }

            }

            // Handle ORDER BY clause

            if (orderByColm != null) {

                orderByIndex = findColumnIndex(headerColumns, orderByColm);

                if (orderByIndex == -1) {

                    throw new IllegalArgumentException("Column not found in ORDER BY clause: " + orderByColm);

                }

            }

            // Load rows into an array

            String[][] rows = new String[100][headerColumns.length];

            int rowCount = 0;

            String line;

            while ((line = reader.readLine()) != null) {

                String[] values = line.split(",(?=(?:[^\"]\*\"[^\"]\*\")\*[^\"]\*$)");

                if (whereCondition == null || values[whereColIndex].equalsIgnoreCase(whereValue)) {

                    rows[rowCount++] = values;

                }

            }

            if (rowCount == 0) {

                System.out.println("No data found in the table.");

                return;

            }

            if (orderByColm != null) {

                sortRows(rows, rowCount, orderByIndex);

            }

            printTable(headerColumns, rows, rowCount);

        }

    }

    public static void handleUpdateCase(String remainingString) throws IOException {

        String lowerCaseString = remainingString.toLowerCase().trim();

        if (!lowerCaseString.contains(" set ") || !lowerCaseString.contains(" having ") || !lowerCaseString.startsWith("table ")) {

            throw new IllegalArgumentException("Invalid UPDATE syntax. Use: UPDATE TABLE table\_name SET column TO value HAVING column = value");

        }

        int setIndex = lowerCaseString.indexOf(" set ");

        int havingIndex = lowerCaseString.indexOf(" having ");

        String tableName = remainingString.substring(lowerCaseString.indexOf("table ") + 6, setIndex).trim();

        if (!satisfiesJavaIdentifierRules(tableName)) {

            throw new IllegalArgumentException("Invalid table name: '" + tableName + "'. It must follow Java identifier rules and cannot be a reserved Java keyword.");

        }

        String setClause = remainingString.substring(setIndex + 5, havingIndex).trim();

        String[] setClauseParts = setClause.split("(?i) TO ", 2);

        if (setClauseParts.length != 2) {

            throw new IllegalArgumentException("Invalid SET clause syntax. Use: column TO value");

        }

        String columnToUpdate = setClauseParts[0].trim();

        String newValue = setClauseParts[1].trim().replaceAll("^\"|\"$", "");

        String havingClause = remainingString.substring(havingIndex + 7).trim();

        String[] havingClauseParts = havingClause.split("=", 2);

        if (havingClauseParts.length != 2) {

            throw new IllegalArgumentException("Invalid HAVING clause syntax. Use: column = value");

        }

        String conditionColumn = havingClauseParts[0].trim();

        String conditionValue = havingClauseParts[1].trim().replaceAll("^\"|\"$", "");

        File file = new File("DBMS\_TABLES\\" + tableName + ".txt");

        if (!file.exists()) {

            throw new FileNotFoundException("Table does not exist: " + tableName);

        }

        File tempFile = new File("DBMS\_TABLES\\" + tableName + "\_temp.txt");

        boolean rowsUpdated = false;

        int rowsUpdatedCount = 0;

        try (BufferedReader reader = new BufferedReader(new FileReader(file));

            PrintWriter writer = new PrintWriter(new FileWriter(tempFile))) {

            String headerLine = reader.readLine();

            if (headerLine == null) {

                throw new IOException("Table is empty or corrupted.");

            }

            writer.println(headerLine);

            String[] headerColumns = headerLine.split(",");

            int updateColumnIndex = findColumnIndex(headerColumns, columnToUpdate);

            int conditionColumnIndex = findColumnIndex(headerColumns, conditionColumn);

            if (updateColumnIndex == -1) {

                throw new IllegalArgumentException("Column not found in SET clause: " + columnToUpdate);

            }

            if (conditionColumnIndex == -1) {

                throw new IllegalArgumentException("Column not found in HAVING clause: " + conditionColumn);

            }

            String line;

            while ((line = reader.readLine()) != null) {

                String[] row = line.split(",(?=(?:[^\"]\*\"[^\"]\*\")\*[^\"]\*$)");

                if (row[conditionColumnIndex].equalsIgnoreCase(conditionValue)) {

                    row[updateColumnIndex] = newValue;

                    rowsUpdated = true;

                    rowsUpdatedCount++;

                }

                writer.println(String.join(",", row));

            }

        } catch (Exception e) {

            // Always delete the temp file in case of an exception

            if (tempFile.exists()) {

                tempFile.delete();

            }

            throw e; // Re-throw the exception after cleanup

        }

        if (rowsUpdated) {

            if (file.delete() && tempFile.renameTo(file)) {

                System.out.println(rowsUpdatedCount + " rows updated successfully in table: " + tableName);

            } else {

                throw new IOException("Error updating the table.");

            }

        } else {

            if (tempFile.exists() && tempFile.delete()) {

                System.out.println("Temporary file deleted successfully.");

            }

            System.out.println("No rows updated. Condition not met.");

        }

    }

    public static void handleDeleteCase(String remainingString) throws IOException {

        String lowerCaseString = remainingString.toLowerCase();

        if (!lowerCaseString.contains(" having ") || !lowerCaseString.startsWith("from table ")) {

            throw new IllegalArgumentException("Invalid DELETE syntax. Use: DELETE FROM TABLE table\_name HAVING column=value");

        }

        int havingIndex = lowerCaseString.indexOf(" having ");

        String tableName = remainingString.substring(0, havingIndex).replaceFirst("(?i)FROM TABLE ", "").trim();

        // Validate table name

        if (!satisfiesJavaIdentifierRules(tableName)) {

            throw new IllegalArgumentException("Invalid table name: '" + tableName + "'. It must follow Java identifier rules and cannot be a reserved Java keyword.");

        }

        String condition = remainingString.substring(havingIndex + 7).trim();

        String[] conditionParts = condition.split("=");

        if (conditionParts.length != 2) {

            throw new IllegalArgumentException("Invalid condition syntax. Use: column=value");

        }

        String columnToMatch = conditionParts[0].trim();

        String valueToMatch = conditionParts[1].replace("\"", "").trim();

        File file = new File("DBMS\_TABLES\\" + tableName + ".txt");

        if (!file.exists()) {

            throw new FileNotFoundException("Table not found: " + tableName);

        }

        File tempFile = new File("DBMS\_TABLES\\temp\_" + tableName + ".txt");

        boolean rowDeleted = false;

        int deletedRowCount = 0;

        try (BufferedReader reader = new BufferedReader(new FileReader(file));

            PrintWriter writer = new PrintWriter(tempFile)) {

            String header = reader.readLine();

            if (header == null) {

                throw new IOException("Table is empty or corrupted. Unable to perform DELETE operation.");

            }

            writer.println(header);

            String[] columns = header.split(",");

            int columnIndex = -1;

            for (int i = 0; i < columns.length; i++) {

                if (columns[i].trim().equalsIgnoreCase(columnToMatch)) {

                    columnIndex = i;

                    break;

                }

            }

            if (columnIndex == -1) {

                throw new IllegalArgumentException("Column not found: " + columnToMatch);

            }

            String row;

            while ((row = reader.readLine()) != null) {

                String[] rowValues = row.split(",");

                if (!rowValues[columnIndex].trim().equalsIgnoreCase(valueToMatch)) {

                    writer.println(row);

                } else {

                    rowDeleted = true;

                    deletedRowCount++;

                }

            }

        } catch (Exception e) {

            // Always delete the temp file in case of an exception

            if (tempFile.exists()) {

                tempFile.delete();

            }

            throw e; // Re-throw the exception after cleanup

        }

        if (!rowDeleted) {

            tempFile.delete();

            System.out.println("No rows matched the condition.");

        } else {

            if (file.delete() && tempFile.renameTo(file)) {

                System.out.println("Deleted " + deletedRowCount + " rows from the table: " + tableName);

            } else {

                throw new IOException("Error updating the table.");

            }

        }

    }

    public static void displayHelp() {

        System.out.println("\nAvailable Commands Are:");

        System.out.println("1) CREATE table\_name HAVING column1, column2, column3,...");

        System.out.println("2) DROP table\_name TABLE");

        System.out.println("3) SHOW TABLES");

        System.out.println("4) INSERT INTO table\_name VALUES (\"value1\", \"value2\", \"value3\", ...)");

        System.out.println("5) SELECT FROM table\_name WHERE column=value ORDER BY column");

        System.out.println("6) UPDATE TABLE table\_name SET column TO value HAVING column = value");

        System.out.println("7) DELETE FROM table\_name WHERE column=value");

        System.out.println("8) HELP");

        System.out.println("9) EXIT");

    }

    //Helper Methods Being Used In Above Methods

    public static boolean satisfiesJavaIdentifierRules(String wordToCheck) {

        if (wordToCheck == null || wordToCheck.isEmpty()) {

            return false;

        }

        if (!Character.isJavaIdentifierStart(wordToCheck.charAt(0))) {

            return false;

        }

        for (int i = 1; i < wordToCheck.length(); i++) {

            if (!Character.isJavaIdentifierPart(wordToCheck.charAt(i))) {

                return false;

            }

        }

        if (isKeyword(wordToCheck)) {

            return false;

        }

        return true;

    }

    public static boolean isKeyword(String word) {

        String[] javaKeywords = {

            "abstract", "assert", "boolean", "break", "byte", "case", "catch", "char", "class", "const", "continue",

            "default", "do", "double", "else", "enum", "extends", "final", "finally", "float", "for", "goto", "if",

            "implements", "import", "instanceof", "int", "interface", "long", "native", "new", "null", "package",

            "private", "protected", "public", "return", "short", "static", "strictfp", "super", "switch", "synchronized",

            "this", "throw", "throws", "transient", "try", "void", "volatile", "while"

        };

        for (int i = 0; i < javaKeywords.length; i++) {

            if (javaKeywords[i].equalsIgnoreCase(word)) {

                return true;

            }

        }

        return false;

    }

    //Used in handleShowCase Method

    public static void listAllTables(String folderName, String indentation) {

        File obj = new File(folderName);

        String[] files = obj.list();

        if (files != null) {

            for (int i = 0; i < files.length; i++) {

                File anotherObj = new File(folderName + "\\" + files[i]);

                if (anotherObj.isFile() && files[i].endsWith(".txt")) {

                    System.out.println(indentation + files[i]);

                } else if (anotherObj.isDirectory()) {

                    listAllTables(folderName + "\\" + files[i], indentation);

                }

            }

        } else {

            System.out.println(indentation + "No tables found.");

        }

    }

    //Used in handleSelectCase Method

    public static void sortRows(String[][] rows, int rowCount, int columnIndex) {

        for (int i = 0; i < rowCount - 1; i++) {

            for (int j = 0; j < rowCount - i - 1; j++) {

                if (rows[j][columnIndex].compareToIgnoreCase(rows[j + 1][columnIndex]) > 0) {

                    String[] temp = rows[j];

                    rows[j] = rows[j + 1];

                    rows[j + 1] = temp;

                }

            }

        }

    }

    public static void printTable(String[] headerColumns, String[][] rows, int rowCount) {

        for (int i = 0; i < headerColumns.length; i++) {

            System.out.printf("%-15s", headerColumns[i]);

        } System.out.println();

        // Print divider line

        for (int i = 0; i < 15 \* headerColumns.length; i++) {

            System.out.print("-");

        } System.out.println();

        // Print rows

        for (int i = 0; i < rowCount; i++) {

            for (int j = 0; j < rows[i].length; j++) {

                System.out.printf("%-15s", rows[i][j]);

            } System.out.println();

        }

    }

    //Used in handleSelectCase and handleUpdateCase

    public static int findColumnIndex(String[] headerColumns, String columnName) {

        for (int i = 0; i < headerColumns.length; i++) {

            if (headerColumns[i].trim().equalsIgnoreCase(columnName)) {

                return i;

            }

        }

        return -1;

    }

}