German University in Cairo
Faculty of Media Engineering and Technology
Spring 2025
Dr. Mohamed Karam

CSEN604 – Data Bases II

Project MS2 Description (Deadline 19/5 at 11:59 pm)

Introduction:

The main objective of this project is to continue the implementation of other functionalities of the DBMS. The main focus will be directed on index behaviors. Alongside the recovery of data. dive deeper into a subset of the components and functionality provided by a DBMS. Moreover, further focus and attention are to be directed toward the assessment and analysis of the behavior and performance of these components.

The main functionalities to be **implemented** for milestone two are:
1. Create the Bitmap index.
2. Insert into the Bitmap index.
3. Select using index.
4. Data recovery (restoring).

These functionalities will be added to the implementation of milestone one. You either use your own implementation, or use the sample solution of milestone one that is uploaded on the CMS.

For the rest of this document, further elaborations regarding requirements, grading, starter code, and simplification assumptions are provided.

1. Simplification Assumptions:

Towards achieving the main objective of this project, some assumptions are made to simplify some tasks, opening more focus room for others.
These assumptions are **as follows:**

1. **The bitmap index implementation:** The implementation of the bitmap is the same of the one taught in the course, for further information recheck the lecture and practice assignment.

2. **Index creation:** The index is only created when the creation function is called, not upon the table creation.

3. **Index updates alongside the table:** The index should be updated (after creation) with every insertion done to the table. *Hint*: the new values should be inserted in both the table and the index.

4. **The loss data will be a whole page:** The recovery should be done of a whole missing page, not few records from a page, there will be a code given to remove the pages, you are not required to implement a delete function.

2. Grading:

Grading for this milestone will be performed using a set of unit tests for the DBMS level, each with equal weight. In other words, the ratio of the correct (green) tests with respect to the total number of test cases is the assigned grade.
There will a test file provided with the starter code, you can use this for the initial implementation steps, the rest of the tests will be published one week after the publishing of the project description.

3. Starter Code:

You must continue your implementation on one of your own of milestone 1 or use the sample solution of the milestone on the CMS. Your implementation should contain the same files from milestone 1, with milestone 2 files, which are:

1. `DBApp.java`
2. `FileManager.java` (You will be given a new updated file to help with the index).
3. `DBAppTestsMS2.java`

You can add any optional files, like:

1. `BitMapIndex.java`

You will need to take the method signatures from section 4.3. (Implementation details -> DBApp.java) and add them all to the DBApp file you will continue using from milestone 1. **DO NOT CHANGE** any method signature when adding to the `DBApp.java`

4. Implementation details:

The project consists of 4 main components:

1. **`DBAppTestsMS2.java:`**
   This file contains the test units for the advanced level, you are not allowed to change anything in this file.

2. **`FileManager.java:`**
   The file will be the same as the one used in milestone one, with two added new functions to aid with the index implementation.
   Those new methods are:

a. **`Public static boolean storeTableIndex(String tableName, String columnName, BitmapIndex b)`**
   This function is responsible for storing the index created on a specific column in the table. The index is saved in the same folder as the table and its pages in the **'Tables'** folder. The index file is saved as the column name + '.db'. It takes the table name and the indexed column name with the index as input and returns a boolean that shows if the index is saved successfully (return True), or not (returns False).

b. **`public static BitmapIndex loadTableIndex(String tableName, String columnName)`**
   This function is responsible for retrieving the saved index file from the **'Tables'** directory. It takes the table name and the indexed column name as input and returns the loaded index.

3. **`DBApp.java:`**
   This file contains the main project implementation, you are required to finish the new implementation of the new added functions that are responsible for the index (creation and selection), and the data recovery. You are **not allowed** to change any attribute name or function signature at all.
   The following is the details of the DBApp class:

   a. **`Public static ArrayList<String []> validateRecords(String tableName)`**
      This function is responsible for checking if there is any missing records from the pages of the table. The missing pages can be: no pages, one page, and multiple pages. It takes the table name and returns an arraylist of the missing records (the ones that were deleted after its page is removed).

   b. **`public static void recoverRecords(String tableName, ArrayList<String[]> missing)`**
      This function is responsible for recovering the missing records, which may be of a single page or multiple pages, and pages from the table. The missing data should be recovered in its old place, do not just insert the missing records again into the end of the table. It takes the list of missing records and the table name, and doesn't return anything, the new table should just be saved on the hard disk and the trace should be updated.

   c. **`public static void createBitMapIndex(String tableName, String colName)`**
      This function is responsible for creating the bitmap index for a specific column of a given table, then save that index to the hard disk. It takes the table name and the column name that needs to be indexed, and returns nothing.

d. **`public static String getValueBits(String tableName, String colName, String value)`**
This function is responsible for returning the bitstream from the index of a specific column of a specific value. The function take the table name, the column name indexed, and the value to get its bitstream (locations). It returns a string containing the bitstream (locations) of that value.

e. **`Public static ArrayList<String []> selectIndex(String tableName, String[] cols, String[] vals)`**
This function is responsible for selecting rows with a specific condition using index. There are 3 cases for the selection:
- **Selection when all columns have indices created on them:** each index on each column will be used to get the result set of its condition, then the results will be **AND**ed together to get the final result.
- **Selection when only one column has an index on it:** The index must be used to get the result that satisfies the condition on that column, then linear conditioning will be done on the result set to get the final result.
- **Selection when multiple but not all columns have indices on:** The result will be an output of the past two combined, the indices will be used to get the result of each condition on the column alone then **AND**ed together to get a result set, then the final conditions will be done linearly to get the final result.
- **Selection with no indices created:** This will do all linear search to output the results.

f. The **'Trace'** of the table should be updated with each new functionality that is added from the new functionalities.

4. The optional classes are used for deep-level implementation and guidance, alongside the test files for these classes.

5. Main method example:

Example of the main used to test the bitmap index:

```
        public static void main(String []args) throws IOException
{
        FileManager.reset();

        String[] cols = {"id","name","major","semester","gpa"};
        createTable("student", cols);
        String[] r1 = {"1", "stud1", "CS", "5", "0.9"};
        insert("student", r1);
```

```
            String[] r2 = {"2", "stud2", "BI", "7", "1.2"};
            insert("student", r2);

            String[] r3 = {"3", "stud3", "CS", "2", "2.4"};
            insert("student", r3);

            createBitMapIndex("student", "gpa");
            createBitMapIndex("student", "major");

            System.out.println("Bitmap of the value of CS from the major index: 
"+getValueBits("student", "major", "CS"));
            System.out.println("Bitmap of the value of 1.2 from the gpa index: 
"+getValueBits("student", "gpa", "1.2"));


            String[] r4 = {"4", "stud4", "CS", "9", "1.2"};
            insert("student", r4);

            String[] r5 = {"5", "stud5", "BI", "4", "3.5"};
            insert("student", r5);

            System.out.println("After new insertions:");
            System.out.println("Bitmap of the value of CS from the major index: 
"+getValueBits("student", "major", "CS"));
            System.out.println("Bitmap of the value of 1.2 from the gpa index: 
"+getValueBits("student", "gpa", "1.2"));


            System.out.println("Output of selection using index when all columns of 
the select conditions are indexed:");
            ArrayList<String[]> result1 = selectIndex("student", new String[] 
{"major","gpa"}, new String[] {"CS","1.2"});
        for (String[] array : result1) {
            for (String str : array) {
                System.out.print(str + " ");
            }
            System.out.println();
        }
            System.out.println("Last trace of the table: "+getLastTrace("student"));
        System.out.println("--------------------------------");

            System.out.println("Output of selection using index when only one column 
of the columns of the select conditions are indexed:");
            ArrayList<String[]> result2 = selectIndex("student", new String[] 
{"major","semester"}, new String[] {"CS","5"});
        for (String[] array : result2) {
            for (String str : array) {
                System.out.print(str + " ");
            }
            System.out.println();
        }
            System.out.println("Last trace of the table: "+getLastTrace("student"));
        System.out.println("--------------------------------");
        System.out.println("Output of selection using index when some of the columns
of the select conditions are indexed:");
             ArrayList<String[]> result3 = selectIndex("student", new String[]
{"major","semester","gpa" }, new String[] {"CS","5", "0.9"});
        for (String[] array : result3) {
            for (String str : array) {
                System.out.print(str + " ");
            }
            System.out.println();
        }
             System.out.println("Last trace of the table: "+getLastTrace("student"));
        System.out.println("--------------------------------");


             System.out.println("Full Trace of the table:");
             System.out.println(getFullTrace("student"));

             System.out.println("--------------------------------");
             System.out.println("The trace of the Tables Folder:");
             System.out.println(FileManager.trace());

       }
```

This is the output of this code snippet:

```
Bitmap of the value of CS from the major index: 101
Bitmap of the value of 1.2 from the gpa index: 010
After new insertions:
Bitmap of the value of CS from the major index: 10110
Bitmap of the value of 1.2 from the gpa index: 01010
Output of selection using index when all columns of the select conditions are
indexed:
4 stud4 CS 9 1.2
Last trace of the table: Select index condition:[major, gpa]->[CS, 1.2], Indexed
columns: [major, gpa], Indexed selection count: 1, Final count: 1, execution time
(mil):3
--------------------------------
Output of selection using index when only one column of the columns of the select
conditions are indexed:
1 stud1 CS 5 0.9
Last trace of the table: Select index condition:[major, semester]->[CS, 5], Indexed
columns: [major], Indexed selection count: 3, Non Indexed: [semester], Final count:
1, execution time (mil):2
--------------------------------
Output of selection using index when some of the columns of the select conditions are
indexed:
1 stud1 CS 5 0.9
Last trace of the table: Select index condition:[major, semester, gpa]->[CS, 5, 0.9],
Indexed columns: [major, gpa], Indexed selection count: 1, Non Indexed: [semester],
Final count: 1, execution time (mil):2
--------------------------------
Full Trace of the table:
```

```
Table created name:student, columnsNames:[id, name, major, semester, gpa]
Inserted: [1, stud1, CS, 5, 0.9], at page number:0, execution time (mil):2
Inserted: [2, stud2, BI, 7, 1.2], at page number:0, execution time (mil):9
Inserted: [3, stud3, CS, 2, 2.4], at page number:1, execution time (mil):2
Index created for column: gpa, execution time (mil):8
Index created for column: major, execution time (mil):2
Inserted: [4, stud4, CS, 9, 1.2], at page number:1, execution time (mil):4
Inserted: [5, stud5, BI, 4, 3.5], at page number:2, execution time (mil):8
Select index condition:[major, gpa]->[CS, 1.2], Indexed columns: [major, gpa],
Indexed selection count: 1, Final count: 1, execution time (mil):3
Select index condition:[major, semester]->[CS, 5], Indexed columns: [major], Indexed
selection count: 3, Non Indexed: [semester], Final count: 1, execution time (mil):2
Select index condition:[major, semester, gpa]->[CS, 5, 0.9], Indexed columns: [major,
gpa], Indexed selection count: 1, Non Indexed: [semester], Final count: 1, execution
time (mil):2
Pages Count: 3, Records Count: 5, Indexed Columns: [gpa, major]
--------------------------------
The trace of the Tables Folder:
Tables{ student{ 0.db 1.db 2.db gpa.db major.db student.db } }
```

Example of the main used to test the validation and recovery:

```
	public static void main(String []args) throws IOException
	{
		FileManager.reset();
		String[] cols = {"id","name","major","semester","gpa"};
		createTable("student", cols);
		String[] r1 = {"1", "stud1", "CS", "5", "0.9"};
		insert("student", r1);

		String[] r2 = {"2", "stud2", "BI", "7", "1.2"};
		insert("student", r2);

		String[] r3 = {"3", "stud3", "CS", "2", "2.4"};
		insert("student", r3);

		String[] r4 = {"4", "stud4", "CS", "9", "1.2"};
		insert("student", r4);

		String[] r5 = {"5", "stud5", "BI", "4", "3.5"};
		insert("student", r5);

		//////// This is the code used to delete pages from the table
		System.out.println("File Manager trace before deleting pages:
"+FileManager.trace());
		String path =
FileManager.class.getResource("FileManager.class").toString();
          File directory = new File(path.substring(6,path.length()-17) +
File.separator
                    + "Tables//student" + File.separator);
          File[] contents = directory.listFiles();
          int[] pageDel = {0,2};
```

```
            for(int i=0;i<pageDel.length;i++)
            {
               contents[pageDel[i]].delete();
            }
            ////////End of deleting pages code

            System.out.println("File Manager trace after deleting pages: 
"+FileManager.trace());
            ArrayList<String[]> tr = validateRecords("student");
               System.out.println("Missing records count: "+tr.size());

               recoverRecords("student", tr);
               System.out.println("--------------------------------");
               System.out.println("Recovering the missing records.");
               tr = validateRecords("student");
               System.out.println("Missing record count: "+tr.size());
               System.out.println("File Manager trace after recovering missing records: 
"+FileManager.trace());
               System.out.println("--------------------------------");
               System.out.println("Full trace of the table: ");
               System.out.println(getFullTrace("student"));
       }
```

This is the output of this code snippet:

```
File Manager trace before deleting pages: Tables{ student{ 0.db 1.db 2.db student.db 
} }
File Manager trace after deleting pages: Tables{ student{ 1.db student.db } }
Missing records count: 3
--------------------------------
Recovering the missing records.
Missing record count: 0
File Manager trace after recovering missing records: Tables{ student{ 0.db 1.db 2.db 
student.db } }
--------------------------------
Full trace of the table:
Table created name:student, columnsNames:[id, name, major, semester, gpa]
Inserted: [1, stud1, CS, 5, 0.9], at page number:0, execution time (mil):3
Inserted: [2, stud2, BI, 7, 1.2], at page number:0, execution time (mil):2
Inserted: [3, stud3, CS, 2, 2.4], at page number:1, execution time (mil):2
Inserted: [4, stud4, CS, 9, 1.2], at page number:1, execution time (mil):2
Inserted: [5, stud5, BI, 4, 3.5], at page number:2, execution time (mil):2
Validating records: 3 records missing.
Recovering 3 records in pages: [0, 2].
Validating records: 0 records missing.
Pages Count: 3, Records Count: 5, Indexed Columns: []
```

6. Team and submission rules:

Please note the following rules for the project:

1. You can continue with your team from milestone 1, or create a new one.

Note that if you change or split the team into new ones, you are not allowed to use the same project of milestone 1, as the same project can't be shared with more than one team, use the sample solution instead).

Using the same code by multiple teams is considered **CHEATING**, and all teams will get a **ZERO.**

Here is a link that contains the list of teams with the team number based on MS1 submissions.

2. Same team rules apply from milestone 1. The maximum number of members is 4, any team with more than 4 members will get a **ZERO**.
3. The submission link will be available on the CMS before the deadline.
4. You are required to submit a zip file containing the following:
   a. A text file containing the names, IDs, Tutorials, and majors of the team members.
   b. The "src" folder of your project, exported as a zip file.
5. You ARE NOT allowed to change any method name given from the starter code.