German University in Cairo

Faculty of Media Engineering and Technology

CSEN604: Database II

Project – Part A

Instructor: Dr. Wael Abouelsaadat

TA: Ali Elhalawaty && Yousra Ayman

Release Date: Feb 7th, 2018

Background

The course project is worth 20%. Part A (this one) is worth 10%. No best policy.

Instructions

• This assignment should be done in teams of SIX (seven not ok!). If you cannot find a partner, contact your TA, and you will be synched with students.

• Submissions to be made electronically via MET website

Description

Overview

In this project, you are going to build a small database engine with support for a BRIN index data structure. The required functionalities are creating/quering an index, and inserting/removing/updating data from the relation which the index is built on.

The items description below is numbered for ease of communication between you and course staff.

User Relations/Tables

1) Each user table will be stored in one or more pages (depending on the size of the table).

2) You are required to use java binary object file (.class) for emulating a page (to avoid having you work with file system pages, which is not the scope of this course).

3) A single tuple will be stored in a separate object inside the binary file.

4) A file has a predetermined maximum number (N) of rows. For example, if a table has 40000 tuples, and N=200, the table will be stored in 200 binary files.

5) Supported types for a column is one of: java.lang.Integer, java.lang.String, java.lang.Double, java.lang.Boolean and java.util.Date

6) Each table should have an additional column beside those specified by the user. The additional column must be called TouchDate and is initialized with the date/time of row insertion and updated with current date/time every time a row is updated.

Pages

7) You need to postpone the loading of a page until the tuples in that page are actually needed. Note that the purpose of using pages is to avoid loading the entire table's content into memory. Hence, it defeats the purpose to load all the pages of all tables upon program startups.

Meta-Data File

8) Each user table has meta data associated with it; number of columns, data type of columns, which columns have indices built on them.

9) You will need to store the meta-data in a text file. This structure should have the following layout: **Table Name, Column Name, Column Type, Key, Indexed**

For example, if a user creates a table/relation CityShop, specifying several attributes with their types, etc… the file will be:

```
Table Name, Column Name, Column Type, Key, Indexed
CityShop, ID, java.lang.Integer, True, False
CityShop, Name, java.lang.String, False, False
CityShop, X, java.lang.Double, False, True
CityShop, Y, java.util.Double, False, True
CityShop, Z, java.lang.Double, False, True
CityShop, Specialization, java.lang.String, False, True
CityShop, Address, java.lang.String, False, false
```

The above meta data teaches that there are 1 table of 7 tuples (ID, Name, X,Y,Z, Specialization, Address). There are 4 BRIN indices created on this table CityShop.

10) For simplicity, you can store the above metadata in a single file called *metadata.csv*. Do not worry about its size in your solution.

BRIN Index

11) You are required to use BRIN index to support answering range queries on the data.

12) You are going to implement your own BRIN Index data structure. Limit that to 2 levels.

13) Once a table is created, you do not need to create an index. An index will be created later on when the user requests that through a method call to createIndex.

14) You should update existing relevant indices when a tuple is inserted/deleted/updated.

15) Upon application startup; to avoid having to scan all tables to build existing indices, you should save the index itself to disk and load it when the application starts next time.

Required Methods/Class

16) Your main class should be called DBApp and should have the following methods/signature;

```
public void init( );    // this does whatever initialization you would like

public void createTable(String strTableName,
                        Hashtable<String,String> htblColNameType )
                                                                throws DBAppException

public void createBRINIndex(String strTableName,
                            String strColName) throws DBAppException

public void insertIntoTable(String strTableName,
                            Hashtable<String,Object>  htblColNameValue)
                                                                throws DBAppException
public void updateTable(String strTableName,
                        String strKey,
                        Hashtable<String,Object> htblColNameValue   )
                                                              throws DBAppException


public void deleteFromTable(String strTableName,
                            Hashtable<String,Object> htblColNameValue)
                                                             throws DBAppException

public Iterator selectFromTable(String strTableName, String strColumnName,
                                Object[] objarrValues,
                                String[] strarrOperators)
                                                  throws DBAppException
```

Here is an example client code that creates a table, creates an index, does few inserts, and a select;

```
String strTableName = "Student";

Hashtable htblColNameType = new Hashtable( );
htblColNameType.put("id", "java.lang.Integer");
htblColNameType.put("name", "java.lang.String");
htblColNameType.put("gpa", "java.lang.double");
createTable( strTableName,htblColNameType );

createBRINIndex( strTableName, "gpa"  );

Hashtable htblColNameValue = new Hashtable( );

htblColNameValue.put("id", new Integer( 2343432 ));
htblColNameValue.put("name", new String("Ahmed Noor" ) );
htblColNameValue.put("gpa", new Double( 0.95 ) );
insertIntoTable( strTableName , htblColNameValue );

htblColNameValue.clear( );
htblColNameValue.put("id", new Integer( 453455 ));
htblColNameValue.put("name", new String("Ahmed Noor" ) );
htblColNameValue.put("gpa", new Double( 0.95 ) );
insertIntoTable( strTableName , htblColNameValue );

htblColNameValue.clear( );
htblColNameValue.put("id", new Integer( 5674567 ));
htblColNameValue.put("name", new String("Dalia Noor" ) );
htblColNameValue.put("gpa", new Double( 1.25 ) );
insertIntoTable( strTableName , htblColNameValue );

htblColNameValue.clear( );
htblColNameValue.put("id", new Integer( 23498 ));
htblColNameValue.put("name", new String("John Noor" ) );
htblColNameValue.put("gpa", new Double( 1.5 ) );
insertIntoTable( strTableName , htblColNameValue );

htblColNameValue.clear( );
htblColNameValue.put("id", new Integer( 78452 ));
htblColNameValue.put("name", new String("Zaky Noor" ) );
htblColNameValue.put("gpa", new Double( 0.88 ) );
insertIntoTable( strTableName , htblColNameValue );

htblColNameValue.clear( );
Object[]objarrValues = new Object[2];
objarrValues[0] = new Double( 0.75 );
objarrValues[1] = new Double( 1.0 );
String[] strarrOperators = new String[2];
strarrOperators[0] = ">=";
strarrOperators[1] = "<";


// following  call to search for 0.75 >= gpa < 1.0
// always assume operators are Anded, i.e. 0.75 >= gpa && gpa < 1.0
Iterator resultSet = selectFromTable(strTableName, "gpa",
                                    objarrValues, strarrOperators );
```

17) For the parameters, the name documents what is being passed – for example htblColNameType is a hashtable with *key* as ColName and *value* is the Type.

18) Operators can either be >, >=, <, or <=. You are not required to support any other SQL operator. You can assume that ranges will be Anded.

19) `DBAppException` is a generic exception to avoid breaking the test cases when they run. You can customize the Exception by passing a different message upon creation.

20) Iterator is java.util.Iterator It is an interface that enables client code to iterate over the results row by row.

21) You should check on the passed types and donot just accept any type – otherwise, your code will crash will invalid input.

Directory Structure

22) Your submission should be a zipped folder containing the following directory structures/files:

- **teamname**
  - **data**
    - metadata.csv
  - **docs**
  - **classes**
    - **teamname**
      - DBApp.class
      - DBAppTest.class
  - **config**
    - DBApp.config
  - **libs**
  - **src**
    - **teamname**
      - DBApp.java
      - DBAppTest.java
  - Makefile

Where

- **teamname** is your team name!
- **data** contains the important metadata.csv which holds meta information about the user created tables. Also, it will store csv files storing user table content, indices, and any other data related files you need to store.
- **docs** contains html files generated by running javadoc on your source code
- **src** is a the parent directory of all your java source files. You should use *teamname* as the parent package of your files. You can add other Java source files you write here.
- **classes** is the parent directory of all your java class files. When you run make all, the java executables should be generated in **classes**.
- **libs** contains any third-party libraries/jar-files/java-classes. Those are the ones you did not write and using in the assignment.
- **config** contains the important configuration *DBApp.properties* which holds a two parameters as key=value pairs

```
MaximumRowsCountinPage = 200
BRINSize = 15
```

Where

```
MaximumRowsCountinPage as the name
     indicates specifies the maximum number
     of rows in a page.
BRINSize specifies the count of values
     that could be stored in a single BRIN
     file.
```

- You can add other parameters to this file as per need.
- Makefile is a make file! supporting ***make all*** and ***make clean*** targets.

Submission Details

• There are small mini-submissions in this assignment as listed below. You are required to make each submission on time/date. Below is a detailed description of each submission.

Submission #	**Date**	**Submission Focus**	**Methods to be implemented (or changed) in DBApp.java**
1	Feb 17th, 2018, 11:50PM	insert, select	`createTable` `insertIntoTable`
2	Feb 23rd, 2018, 11:50PM	update, delete	`updateTable` `deleteFromTable`
3	Mar 3rd, 2018, 11:50PM	BRIN Index integration	`createIndex` `insertIntoTable` `selectFromTable` `deleteFromTable` `updateTable`

Tips

• *DBApp.properties* file could be read using java.util.Properties class

• For reading a CSV file, java.io.BufferedReader and java.util.StringTokenizer are useful classes.

• You can save any Java object by implementing the java.io.Serializable interface. You don't actually add any code to your class. For more info, check these:
https://www.tutorialspoint.com/java/java_serialization.htm

• You can (but not required to) use reflection to load the data type and also value of a column, for example:

```
strColType  = "java.lang.Integer";
strColValue = "100";
Class class = Class.forName( strColType );
Constructor constructor = class.getConstructor( ….);
… = constructor.newInstance( );
```

For more info on reflection, check this article:
http://download.oracle.com/javase/tutorial/reflect/