Fall 2010 CSCD43
Database Systems Technology
Assignment 2

(Worth 15% of your final mark)

Due Date: Saturday, November 13th, 11:59PM

Description

Overview

In this assignment, you are going to build a simple database engine in Java. The required functionalities are creating tables, building indices, storing data, deleting data, and querying.

User Tables

- Each user table will be stored in pages.
- You are required to use csv (comma separated values) text file for emulating a page.
- A single tuple will be stored in a separate row with commas separating the values.
- A csv 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 20 csv files.
- Deletion of data will result in fragmentation in pages; do not worry about that in your solution.

Pages

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. You need to postpone the loading of a page until the tuples in that page are actually needed for further processing.

MetaData File

- Each user table has meta data associated with it; number of columns, data type of columns, which column is a key, references between table columns and columns in other tables (to enforce constraints),…
- You will need to store the meta data in a file/data-structure. This structure should have the following layout:

Table Name, Column Name, Column Type, Key, Indexed, References

For example, if a user creates two tables, Employee and Department, specifying several attributes with their types, etc… the file will be:

```
Table Name, Column Name, Column Type, Key, Indexed, References
Employee, ID, java.lang.Integer, True, True, null
Employee, Name, java.lang.String, False, False, null,
Employee, Dept, java.lang.String, False, False, Department.ID
Employee, Start_Date, java.util.Date, False, False, null
Department, ID, java.lang.Integer, True, True, null
```

Department, Name, java.lang.String, False, False, null
Department, Location, java.lang.String, False, False, null

- For simplicity, you can store this structure in a single file called *metadata.csv*. Do not worry about its size in your solution.

Indices

- You are required to use B+ trees to support creating primary and secondary *dense* indices. Note that a B+ tree stores in its leafs pointers (handles in Java terminology) to the data objects.
- You can use any available open source B+ tree.
- Once a table is created, you need to create a primary index on the key of that table.
- You should update existing relevant B+ trees when a tuple is inserted/deleted.
- If a secondary index is created after a table has been populated, you have no option but to scan the whole table.
- 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.
- If a select is executed (by calling `selectFromTable` method below), and a column is referenced in the select that has been already indexed, then you should search in the index.

Main App Class

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

```
public void init( );

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

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

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

public void deleteFromTable(String strTableName,
                      Hashtable<String,String> htblColNameValue,
                      String strOperator)
                                throws DBEngineException

public Iterator selectFromTable(String strTable,
                      Hashtable<String,String> htblColNameValue,
                      String strOperator)
                                throws DBEngineException

public void saveAll( ) throws DBEngineException
```

```
// Notes:
//         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.
//
//         strOperator can either be OR or AND
//
//         DBEngineException 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.
//
//         Iterator is java.util.Iterator It is an interface that
//         enables client code to iterate over the results row
//         by row. Do not implement the remove method.
//
//         The method saveAll saves all data pages and indices to
//         disk and can be called at any time (not just before
//         the program terminates)
```

Directory Structure

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

```
teamname-A2
    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 two parameters as key=value pairs

```
MaximumRowsCountinPage = 200
BPlusTreeN = 20
```

  Where

```
MaximumRowsCountinPage as the name
     indicates specifies the maximum number
     of rows in a csv/page.
BPlusTreeN specifies the count of values
     that could be stored in a B+ tree node
```

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

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.

• Check the following references for design patterns for loading pages only when needed:
http://en.wikipedia.org/wiki/Proxy_pattern http://en.wikipedia.org/wiki/Lazy_loading

• 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 this article:
http://java.sun.com/developer/technicalArticles/Programming/serialization/

• 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/

• Note that because you are using dense indices, the location of where to actually insert a tuple in a table does not matter as the B+ tree will refer to the page/line of that tuple. Neither it matters whether the table/page is sorted or not.

• Deletion of a tuple has to be managed carefully. If you delete the line containing the tuple, it will affect the location of lines which follows in that page. Hence, it is better to use a tomb stone to indicate that the tuple has been deleted without actually removing the line.

• When you are inserting, it is safer (but more expensive) to always insert into the saved page directly. This way if your program terminates suddenly (e.g power goes off), the loss is minimized. Do not be tempted into inserting into the memory data structure and postponing the saving to disk to later. If you are going to do that – use a timer thread that continuously callbacks part of your code to save the data structures from memory to disk.

Instructions

• Make sure to read the following sections in the course info sheet: Course website, Assignments and late submissions, Remarking requests, Email, and Academic Offences.

• This assignment should be done in teams of TWO

• Assignment to be submitted electronically via http://portal.utoronto.ca/