German University in Cairo

German University in Cairo
Faculty of Media Engineering and Technology

Database II Assignment 3

Dr. Wael Abouelsaadat

Release Date: May 3rdh, 2014
Due Date: May 22nd, 2014

Introduction

In this assignment, you are going to extend your A1 solution by adding more components which typically exist in a DBMS. You will be adding a buffer manager, a log manager, a transaction manager, and a recovery manager. Notes on the new classes follow below. You are free to add helper classes as deemed required. Though your classes must have the specified method signatures as below, you can add attributes and methods as required.

Instructions

• This assignment should be done in teams of SIX. If you cannot find a team of 6, contact your TA: fadwa.elhussini@guc.edu.eg. She will synch you with other students.
• Assignment to be submitted electronically via MET website
• Note: you should be able to start in this assignment ASAP.

[8%] SQL Processing

You are going to support processing SQL commands by using the following library; http://www.sqlparser.com/
The library parses an input SQL statement and outputs a parse tree which you can traverse to find the statements in the SQL.

[2%] Command Line Tool

You are going to support a command line interface by using the following library; http://jline.sourceforge.net/
The library enables you to present a professional looking command line interface.

[20%] BufferManager

```
// BufferManager manages the reading and writing of pages from
// disk. It maintains two lists: list of empty slots, and list of
// used slots. DBApp.config has two new parameters
// (MinimumEmptyBufferSlots/MaximumUsedBufferSlots) to
// indicate the min and max size of the lists.
// Upon requesting a page via read(…), the BufferManager will
// load it from disk if it is not already in memory and look for
// an empty slot to store it in. The slot is moved from empty
// list and added to used list.
// BufferManager should have an internal thread that runs
// every 2 minutes to remove the least recently used (LRU) page
// from memory. LRU is described on page 748 of the textbook and
// in lecture slides. If the page has changed since loaded from
// disk(dirty), then write it to disk, otherwise only delete from
// memory.
// Note that if the empty list reaches the minimum, or the full
// list reaches maximum, then you must run the LRU algorithm
// immediately.
// BufferManager should always return a copy of the page
// when the read method is called.
// BufferManager.read(…) method will accept a flag indicating
// whether the page is being read for modification or not.
// PageID and Page classes are left for you to define.

// methods
public void init( );

public synchronized void read(  PageID pageID, Page page,
                                boolean bModify );

public synchronized void write( PageID pageID, Page page );
```

[20%] LogManager

```
// LogManager is class shared by all running Transactions.
// It implements an undo/redo logging mechanism. The location
// of the log file is to be loaded from the DBApp.config file
// strTransID is a unique identifier for a transaction. You can
// use System.currentTimeMillis() concatenated with a random
// generated number to create such unique ID.

// methods

public void init( );

public synchronized void flushLog( );

public synchronized void recordStart( String strTransID );

public synchronized void recordUpdate(  String strTransID,
                                        PageID page,
                                           String strKeyValue,
                                           String strColName,
                                           Object objOld,
                                           Object objNew);

    public synchronized void recordInsert(String strTransID,
                                          PageID page,
                             Hashtable<String,String> htblColValues);

    public synchronized void recordDelete(String strTransID,
                                          PageID page,
                                          String strKeyValue,
                             Hashtable<String,String> htblColValues);

    public synchronized void recordCommit( String strTransID );
```

[10%] RecoveryManager

```
    // RecoveryManager.recover() is run every time the DBMS
    // is started. It implements an undo/redo recovery policy.
    // The location of the log file must be loaded from the
    // file DBApp.config

    // methods
    public void recover( );
```

[10%] Transaction

```
    // Transaction class runs in its own thread to execute
    // a sequence of steps.
    // A transaction object is not reusable, i.e. every incoming SQL
    // statement will result in the creation of a new transaction
    // object.
    // A transaction is responsible for calling the BufferManager
    // to read and write pages and also responsible for calling the
    // LogManager to record the steps being executed.
    // The execute method starts the thread associated with
    // the Transaction to run the steps.
    // When a transaction ends, make sure to clear all it's
    // attributes by setting them to null. This will help Java
    // garbage collector to identify those objects as being unused
    // and removes them from memory faster.

    // methods
    public void init( BufferManager bufManager,
                      LogManager logManager,
                      Vector<Steps> vSteps);

    public void execute( );
```

***[10%]* Step**
```
    // Step is an abstract class from which inherits all the
    // concrete steps. A transaction is a sequence of concrete steps.

    // methods
    // left for you to design and implement
```

PageRead *extends* Step
```
    // A transaction needs to read a page from BufferManager.
    // This step encapsulates such an activity.

    // methods
    // left for you to design and implement
```

PageWrite *extends* Step
```
    // A transaction might need to write a page to BufferManager.
    // This step encapsulates such an activity.

    // methods
    // left for you to design and implement
```

TupleUpdate *extends* Step
```
    // A transaction might need to modify an existing value in a page
    // This step encapsulates such an activity.

    // methods
    // left for you to design and implement
```

TupleInsert *extends* Step
```
    // A transaction might need to insert a new value in a page.
    // This step encapsulates such an activity.

    // methods
    // left for you to design and implement
```

TupleDelete *extends* Step
```
    // A transaction might need to delete a tuple in a page.
    // This step encapsulates such an activity.

    // methods
    // left for you to design and implement
```

***[20%]* TransactionManager**
```
    // TransactionManager will contain the indices B+ tree(s).
    //  TransactionManager is responsible for planning the execution
    // of an SQL. For this assignment, you are going to implement
```

```
// that in the form of generating a sequence of steps using the
// steps concrete classes mentioned above. The A1 implementation
// of the three methods insertIntoTable, deleteFromTable and
// selectFromTable will be partially substituted by those
// concrete steps. The new method updateTable is similar to them.
// You still have to use the B+ tree if there is one for the
// column you are processing.  However, you cannot read/write
// directly to disk as you did in A1. You will have to create a
// transaction object, initialize it with the steps, and run the
// transaction object in it's own thread. The transaction object
// will in turn talk to the buffer manager to read/write the
// pages.
// The createTable and createIndex should work as before. You can
// still write to metadata.csv from TransactionManager to
// document the creation of tables and indices.

// methods

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 updateTable(String strTableName,
                      Hashtable<String,String> htblColNameValue)
                     throws DBAppException

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

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

public void saveAll( ) throws DBEngineException
```

Directory Structure

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

```
teamname-A3
    data
        metadata.csv
        log.csv
    docs
    classes
        teamname
            DBApp.class
    config
        DBApp.config
    libs
    src
        teamname
            DBApp.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
MinimumEmptyBufferSlots = 1
MaximumUsedBufferSlots  = 40
LogfilePath= ../../data/log.csv
```

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
MinimumEmptyBufferSlots
      and MaximumUsedBufferSlots   indicate
      the minimum and maximum size of the
```

```
                    internal   data   structures   inside
                    BufferManager
                LogfilePath specifies the relative path to
                    the log file.
```

You can add other parameters to this file as per need.

- Makefile is a make file! supporting ***make all*** and ***make clean*** targets.