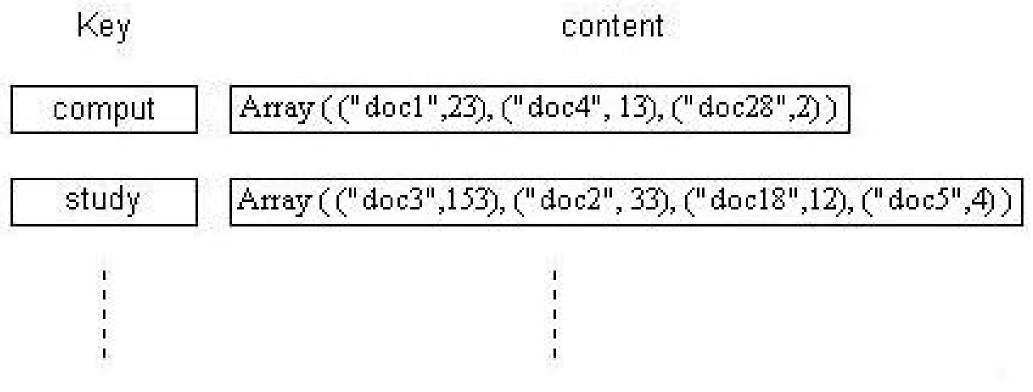
# COMP4321 Lab 1 - JDBM

## 1 Introduction

JDBM is a transactional persistence engine for Java. It provides the basic database functions based on scalable data structures: hash table and B+ Tree. JDBM provides functions for storing, fetching, searching and deleting records according to the specified keys. You need not write your own hashing functions or build search trees to accomplish the search function. JDBM can do these for you.

In JDBM, the basic tuple is a key/content pair. Since the key and the content are of java.lang.Object type, we can store any variables into the database without complex conversions, which are required when using other database managers. We can also store an array into the value part of the triple so that we do not need to create our own method to separate the postings.

Maybe the previous description is a little bit abstract and confusing. Don't worry, let us take an example to see what JDBM can do and why it is convenient for us to build the search engine. As what we learned in the course, search engine has to index the pages and keywords. After indexing, we have to store it in the database to assure the indexed pages/keywords are still there when we restart the computer. What will happen if we don't store it on database? It means all the indexed results will be removed from the memory when the server is down, which causes search engine to crawl and index all the pages again. When the number of pages is small (e.g. in the first phase of our project, only 30 pages are needed to be crawled and indexed) it is OK without database, but how about billions of pages, like what Google deals with? We can view JDBM as a kind of database. It can store the data in like other database (e.g.SQL Server and MySQL). However, the difference is, the concept of "table" in traditional database system is "HTree" or "BTree"in RecordManager. HTree and BTree in JDBM are the implementation of hashtable and B+Tree, which are the abstract data structure. We don't need to care the implementation details and the structures of them, however, what we need to know is the so called HTree/BTree can help us store data in a structure of hashtable/B+Tree. We can add/remove/search a particular entry in HTree/BTree (implementation of hashtable/B+Tree), which can help us index the pages and store the indexed results into hard disk. The figure below is an example of how to index the keywords and the structure of hashtable.


Key
content
comput
Array(("doc1",23), ("doc4", 13), ("doc28",2))
study
Array(("doc3",153), ("doc2", 33), ("doc18",12), ("doc5",4))

In JDBM, more than one relation (HTree/BTree) can be stored in a RecordManager, which is associcated to a db file. There is also a lg file to store the transaction log. The RecordManager can store two types of objects: jdbm.btree(B+ tree) and jdbm.htree(hash table). We can create a B+ tree or hash table based on our need and store it in a RecordManager.

## 2 Specification

Some commonly used classes

1. RecordManager
2. RecordManagerFactory
3. btree
4. htree
5. FastIterator
6. TupleBrowser

The complete javadoc can be found in http://jdbm.sourceforge.net/V1.0/doc/api or the downloaded jdbm-1.0.zip

## 3 Sample Programs

Here is a simple example using hash table.

```java
import jdbm.RecordManager;
import jdbm.RecordManagerFactory;
import jdbm.htree.HTree;
import jdbm.helper.FastIterator;

public class JDBMsample
{
    public static void main(String[] args)
    {
        try
        {
            RecordManager recman;
            HTree hashtable;
               // Create a RecordManager name "testRM"
               recman = RecordManagerFactory.createRecordManager("testRM");
               // get the record id of the object named "ht1"
               long recid = recman.getNamedObject("ht1");

               if (recid != 0)

               {
                   // load the hash table named"ht1"from the RecordManager
                   hashtable = HTree.load(recman, recid);
               }
               else
               {
                   // create a hash table in the RecordManager
                   hashtable = HTree.createInstance(recman);
                   // set the name of the hash table to "ht1"
                   recman.setNamedObject( "ht1", hashtable.getRecid() );
               }

               // add some triples into the hash table
               hashtable.put("key1", "context 1");
               hashtable.put("key2", "context 2");
               hashtable.put("key3", "context 3");
               hashtable.put("key4", "context 4");

               // get the content of the "key3" from the hash table
               System.out.println( hashtable.get("key3"));
```

```
              // remove the triple with key = "key2"
              hashtable.remove( "key2" );

              // iterate through all keys
              FastIterator iter = hashtable.keys();

              String key;
              while( (key = (String)iter.next())!=null)
              {
                  // get and print the content of each key
                  System.out.println(key + " : " + hashtable.get(key));
              }

              // commit the changes
              recman.commit();

              // close the RecordManager
              recman.close();
              }
              catch(java.io.IOException ex)
              {
                   System.err.println(ex.toString());
              }
       }
}
```

---

To compile and execute the programs, we must firstly download the library from http://prdownloads.sourceforge.net/jdbm/jdbm-1.0.zip?download.

Then we can extract the jdbm-1.0.jar from the jdbm-1.0.zip.

If the jdbm-1.0.jar is placed in "`~/lib`", the compile command for the JDBMsample.java is "`javac -cp ~/lib/jdbm-1.0.`

and the program is executed by "`java -cp ~/lib/jdbm-1.0.jar:. JDBMsample`". (Note there has a space in front of "JDBMsample", and there should not have a space after jdbm-1.0.jar, and for Windows, replace the colon with semicolon)

Here are two official examples:
FamousPeople.java
FruitBasket.java

You are asked to implement a database similar to that illustrated in the figure above.

## 4 Exercise

Requirements
The key is a single word, the content is a list of "docX Y", where X and Y are any distinct integers. An example of a (key, content) pair is: ("dog", "doc23 43 doc10 6"). The filename of the db file is "lab1.db". In your program, you are expected to mainly implement three functions:

1. public static void addEntry(String word, int x, int y, HTree hashtable);// Add a "docX Y" entry for the key "word"
For example:
- Before, one of the (key, content) pairs in the database is:
("dog", "doc23 43 doc10 6")
- After calling addEntry("dog", 5, 9):
("dog", "doc23 43 doc10 6 doc5 9")

2. public static void delEntry(String word); // Delete the word and its list from the database
For example, after calling delEntry("dog", hashtable), the key "dog" and its content are deleted from

the database

3. public static void printAll();// Print all the data in the database
For example, a call for printAll(hashtable) may produce the following output:

```
dog = doc23 43 doc10 6
cat = doc4 2
pig = doc98 20 doc65 22 doc19 45
```

Start your work on the skeleton program InvertedIndex.java. The following output is expected:

```
First print
cat = doc2 6
dog = doc1 33
Second print
cat = doc2 6 doc8 3 doc11 106
dog = doc1 33 doc6 73 doc8 83 doc10 5
Third print
cat = doc2 6 doc8 3 doc11 106
```

Submission
no need to submit

## 5 Useful Links

1. The official site of jdbm: http://jdbm.sourceforge.net/
2. The javadoc of jdbm: http://jdbm.sourceforge.net/V1.0/doc/api/
3. The java API: http://download.oracle.com/javase/1.4.2/docs/api/index.html