LUCENE - DELETE DOCUMENT OPERATION

Delete document is another important operation as part of indexing process.This operation is used when already indexed contents are updated and indexes become invalid or indexes become very large in size then in order to reduce the size and update the index, delete operations are carried out.

We delete *Documents* containing *Fields* to *IndexWriter* where IndexWriter is used to update indexes.

Now we'll show you a step by step process to get a kick start in understanding of delete document using a basic example.

Delete a document from an index.

- Create a method to delete a lucene document of an obsolete text file.

```
private void deleteDocument(File file) throws IOException{

   //delete indexes for a file
   writer.deleteDocument(new Term(LuceneConstants.FILE_NAME,file.getName()));

   writer.commit();
   System.out.println("index contains deleted files: "+writer.hasDeletions());
   System.out.println("index contains documents: "+writer.maxDoc());
   System.out.println("index contains deleted documents: "+writer.numDoc());
}
```

Create a IndexWriter

- IndexWriter class acts as a core component which creates/updates indexes during indexing process.
- Create object of IndexWriter.
-
- Create a lucene directory which should point to location where indexes are to be stored.
-
- Initialize the IndexWricrter object created with the index directory, a standard analyzer having version information and other required/optional parameters.

```
private IndexWriter writer;

public Indexer(String indexDirectoryPath) throws IOException{
   //this directory will contain the indexes
   Directory indexDirectory =
      FSDirectory.open(new File(indexDirectoryPath));
   //create the indexer
   writer = new IndexWriter(indexDirectory,
      new StandardAnalyzer(Version.LUCENE_36),true,
      IndexWriter.MaxFieldLength.UNLIMITED);
}
```

Delete document and start reindexing process

Following two are the ways to delete the document.

- **deleteDocuments*Term*** - Delete all the documents containing the term.
- **deleteDocuments*Term*[]** - Delete all the documents containing any of the terms in the array.

- **deleteDocuments*Query*** - Delete all the documents matching the query.
- **deleteDocuments*Query*[]** - Delete all the documents matching the query in the array.
- **deleteAll** - Delete all the documents.

```
private void indexFile(File file) throws IOException{
   System.out.println("Deleting index for "+file.getCanonicalPath());
   deleteDocument(file);
}
```

Example Application

Let us create a test Lucene application to test indexing process.

Step	Description
1	Create a project with a name *LuceneFirstApplication* under a package *com.tutorialspoint.lucene* as explained in the *Lucene - First Application* chapter. You can also use the project created in *EJB - First Application* chapter as such for this chapter to understand indexing process.
2	Create *LuceneConstants.java*,*TextFileFilter.java* and *Indexer.java* as explained in the *Lucene - First Application* chapter. Keep rest of the files unchanged.
3	Create *LuceneTester.java* as mentioned below.
4	Clean and Build the application to make sure business logic is working as per the requirements.

LuceneConstants.java

This class is used to provide various constants to be used across the sample application.

```
package com.tutorialspoint.lucene;

public class LuceneConstants {
   public static final String CONTENTS="contents";
   public static final String FILE_NAME="filename";
   public static final String FILE_PATH="filepath";
   public static final int MAX_SEARCH = 10;
}
```

TextFileFilter.java

This class is used as a .txt file filter.

```
package com.tutorialspoint.lucene;

import java.io.File;
import java.io.FileFilter;

public class TextFileFilter implements FileFilter {

   @Override
   public boolean accept(File pathname) {
      return pathname.getName().toLowerCase().endsWith(".txt");
   }
}
```

Indexer.java

This class is used to index the raw data so that we can make it searchable using lucene library.

```java
package com.tutorialspoint.lucene;

import java.io.File;
import java.io.FileFilter;
import java.io.FileReader;
import java.io.IOException;

import org.apache.lucene.analysis.standard.StandardAnalyzer;
import org.apache.lucene.document.Document;
import org.apache.lucene.document.Field;
import org.apache.lucene.index.CorruptIndexException;
import org.apache.lucene.index.IndexWriter;
import org.apache.lucene.index.Term;
import org.apache.lucene.store.Directory;
import org.apache.lucene.store.FSDirectory;
import org.apache.lucene.util.Version;

public class Indexer {

   private IndexWriter writer;

   public Indexer(String indexDirectoryPath) throws IOException{
      //this directory will contain the indexes
      Directory indexDirectory =
         FSDirectory.open(new File(indexDirectoryPath));

      //create the indexer
      writer = new IndexWriter(indexDirectory,
         new StandardAnalyzer(Version.LUCENE_36),true,
         IndexWriter.MaxFieldLength.UNLIMITED);
   }

   public void close() throws CorruptIndexException, IOException{
      writer.close();
   }

   private void deleteDocument(File file) throws IOException{

      //delete indexes for a file
      writer.deleteDocuments(
         new Term(LuceneConstants.FILE_NAME,file.getName()));

   writer.commit();
   }

   private void indexFile(File file) throws IOException{
      System.out.println("Deleting index: "+file.getCanonicalPath());
      deleteDocument(file);
   }

   public int createIndex(String dataDirPath, FileFilter filter)
      throws IOException{
      //get all files in the data directory
      File[] files = new File(dataDirPath).listFiles();

      for (File file : files) {
         if(!file.isDirectory()
            && !file.isHidden()
            && file.exists()
            && file.canRead()
            && filter.accept(file)
         ){
            indexFile(file);
         }
      }
      return writer.numDocs();
   }
}
```

LuceneTester.java

This class is used to test the indexing capability of lucene library.

```
package com.tutorialspoint.lucene;

import java.io.IOException;

public class LuceneTester {

   String indexDir = "E:\\Lucene\\Index";
   String dataDir = "E:\\Lucene\\Data";
   Indexer indexer;

   public static void main(String[] args) {
      LuceneTester tester;
      try {
         tester = new LuceneTester();
         tester.createIndex();
      } catch (IOException e) {
         e.printStackTrace();
      }
   }

   private void createIndex() throws IOException{
      indexer = new Indexer(indexDir);
      int numIndexed;
      long startTime = System.currentTimeMillis();
      numIndexed = indexer.createIndex(dataDir, new TextFileFilter());
      long endTime = System.currentTimeMillis();
      indexer.close();
   }
}
```

Data & Index directory creation

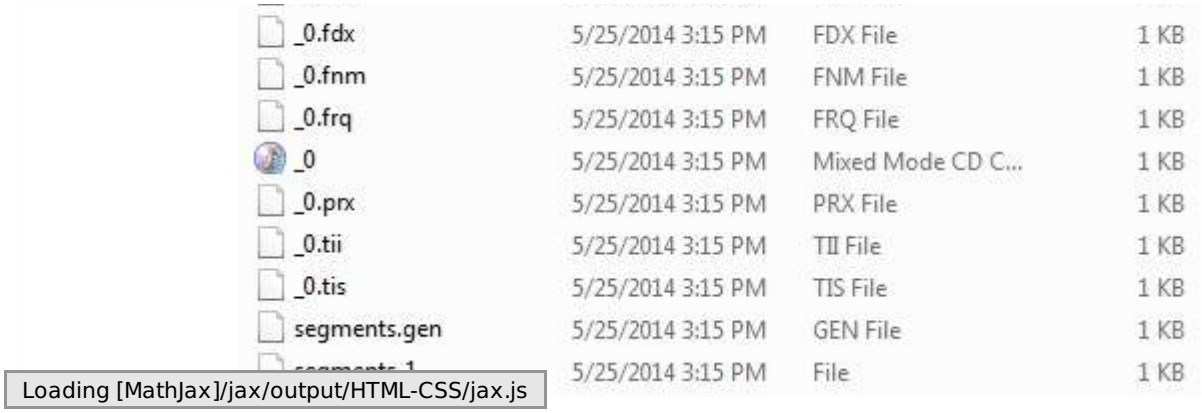
I've used 10 text files named from record1.txt to record10.txt containing simply names and other details of the students and put them in the directory **E:\Lucene\Data.** Test Data. An index directory path should be created as **E:\Lucene\Index**. After running this program, you can see the list of index files created in that folder.

Running the Program:

Once you are done with creating source, creating the raw data, data directory and index directory, you are ready for this step which is compiling and running your program. To do this, Keep LuceneTester.Java file tab active and use either **Run** option available in the Eclipse IDE or use **Ctrl + F11** to compile and run your **LuceneTester** application. If everything is fine with your application, this will print the following message in Eclipse IDE's console:

```
Deleting index E:\Lucene\Data\record1.txt
Deleting index E:\Lucene\Data\record10.txt
Deleting index E:\Lucene\Data\record2.txt
Deleting index E:\Lucene\Data\record3.txt
Deleting index E:\Lucene\Data\record4.txt
Deleting index E:\Lucene\Data\record5.txt
Deleting index E:\Lucene\Data\record6.txt
Deleting index E:\Lucene\Data\record7.txt
Deleting index E:\Lucene\Data\record8.txt
Deleting index E:\Lucene\Data\record9.txt
10 File indexed, time taken: 109 ms
```

Once you've run the program successfully, you will have following content in your **index directory:**



_0.fdx	5/25/2014 3:15 PM	FDX File	1 KB
_0.fnm	5/25/2014 3:15 PM	FNM File	1 KB
_0.frq	5/25/2014 3:15 PM	FRQ File	1 KB
_0	5/25/2014 3:15 PM	Mixed Mode CD C...	1 KB
_0.prx	5/25/2014 3:15 PM	PRX File	1 KB
_0.tii	5/25/2014 3:15 PM	TII File	1 KB
_0.tis	5/25/2014 3:15 PM	TIS File	1 KB
segments.gen	5/25/2014 3:15 PM	GEN File	1 KB
segments.1	5/25/2014 3:15 PM	File	1 KB