package com.twitter.search.core.earlybird.index.inverted;

import java.io.IOException;

import org.apache.lucene.index.Terms;

import org.apache.lucene.index.TermsEnum;

import org.apache.lucene.util.BytesRef;

import com.twitter.search.common.schema.base.EarlybirdFieldType;

import com.twitter.search.common.util.io.flushable.Flushable;

import com.twitter.search.core.earlybird.facets.FacetLabelProvider;

import com.twitter.search.core.earlybird.index.EarlybirdIndexSegmentAtomicReader;

/\*\*

\* Inverted index for a single field.

\*

\* Example: The field is "hashtags", this index contains a mapping from all the hashtags

\* that we've seen to a list of postings.

\*/

public abstract class InvertedIndex implements FacetLabelProvider, Flushable {

protected final EarlybirdFieldType fieldType;

public InvertedIndex(EarlybirdFieldType fieldType) {

this.fieldType = fieldType;

}

public EarlybirdFieldType getFieldType() {

return fieldType;

}

/\*\*

\* Get the internal doc id of the oldest doc that includes term.

\* @param term the term to look for.

\* @return The internal docid, or TERM\_NOT\_FOUND.

\*/

public final int getLargestDocIDForTerm(BytesRef term) throws IOException {

final int termID = lookupTerm(term);

return getLargestDocIDForTerm(termID);

}

/\*\*

\* Get the document frequency for this term.

\* @param term the term to look for.

\* @return The document frequency of this term in the index.

\*/

public final int getDF(BytesRef term) throws IOException {

final int termID = lookupTerm(term);

if (termID == EarlybirdIndexSegmentAtomicReader.TERM\_NOT\_FOUND) {

return 0;

}

return getDF(termID);

}

public boolean hasMaxPublishedPointer() {

return false;

}

public int getMaxPublishedPointer() {

return -1;

}

/\*\*

\* Create the Lucene magic Terms accessor.

\* @param maxPublishedPointer used by the skip list to enable atomic document updates.

\* @return a new Terms object.

\*/

public abstract Terms createTerms(int maxPublishedPointer);

/\*\*

\* Create the Lucene magic TermsEnum accessor.

\* @param maxPublishedPointer used by the skip list to enable atomic document updates.

\* @return a new TermsEnum object.

\*/

public abstract TermsEnum createTermsEnum(int maxPublishedPointer);

/\*\*

\* Returns the number of distinct terms in this inverted index.

\* For example, if the indexed documents are:

\* "i love chocolate and i love cakes"

\* "i love cookies"

\*

\* then this method will return 6, because there are 6 distinct terms:

\* i, love, chocolate, and, cakes, cookies

\*/

public abstract int getNumTerms();

/\*\*

\* Returns the number of distinct documents in this index.

\*/

public abstract int getNumDocs();

/\*\*

\* Returns the total number of postings in this inverted index.

\*

\* For example, if the indexed documents are:

\* "i love chocolate and i love cakes"

\* "i love cookies"

\*

\* then this method will return 10, because there's a total of 10 words in these 2 documents.

\*/

public abstract int getSumTotalTermFreq();

/\*\*

\* Returns the sum of the number of documents for each term in this index.

\*

\* For example, if the indexed documents are:

\* "i love chocolate and i love cakes"

\* "i love cookies"

\*

\* then this method will return 8, because there are:

\* 2 documents for term "i" (it doesn't matter that the first document has the term "i" twice)

\* 2 documents for term "love" (same reason)

\* 1 document for terms "chocolate", "and", "cakes", "cookies"

\*/

public abstract int getSumTermDocFreq();

/\*\*

\* Lookup a term.

\* @param term the term to lookup.

\* @return the term ID for this term.

\*/

public abstract int lookupTerm(BytesRef term) throws IOException;

/\*\*

\* Get the text for a given termID.

\* @param termID the term id

\* @param text a BytesRef that will be modified to contain the text of this termid.

\*/

public abstract void getTerm(int termID, BytesRef text);

/\*\*

\* Get the internal doc id of the oldest doc that includes this term.

\* @param termID The termID of the term.

\* @return The internal docid, or TERM\_NOT\_FOUND.

\*/

public abstract int getLargestDocIDForTerm(int termID) throws IOException;

/\*\*

\* Get the document frequency for a given termID

\* @param termID the term id

\* @return the document frequency of this term in this index.

\*/

public abstract int getDF(int termID);

}