package com.twitter.search.earlybird.search.queries;

import java.io.IOException;

import org.apache.lucene.index.LeafReader;

import org.apache.lucene.index.LeafReaderContext;

import org.apache.lucene.index.NumericDocValues;

import org.apache.lucene.search.BooleanClause;

import org.apache.lucene.search.BooleanQuery;

import org.apache.lucene.search.DocIdSetIterator;

import org.apache.lucene.search.IndexSearcher;

import org.apache.lucene.search.Query;

import org.apache.lucene.search.ScoreMode;

import org.apache.lucene.search.Weight;

import com.twitter.search.common.query.DefaultFilterWeight;

import com.twitter.search.common.schema.earlybird.EarlybirdFieldConstants.EarlybirdFieldConstant;

import com.twitter.search.core.earlybird.index.util.AllDocsIterator;

import com.twitter.search.core.earlybird.index.util.RangeFilterDISI;

import com.twitter.search.earlybird.common.userupdates.UserTable;

public final class UserFlagsExcludeFilter extends Query {

/\*\*

\* Returns a query that filters hits based on their author flags.

\*

\* @param excludeAntisocial Determines if the filter should exclude hits from antisocial users.

\* @param excludeOffensive Determines if the filter should exclude hits from offensive users.

\* @param excludeProtected Determines if the filter should exclude hits from protected users

\* @return A query that filters hits based on their author flags.

\*/

public static Query getUserFlagsExcludeFilter(UserTable userTable,

boolean excludeAntisocial,

boolean excludeOffensive,

boolean excludeProtected) {

return new BooleanQuery.Builder()

.add(new UserFlagsExcludeFilter(

userTable, excludeAntisocial, excludeOffensive, excludeProtected),

BooleanClause.Occur.FILTER)

.build();

}

private final UserTable userTable;

private final boolean excludeAntisocial;

private final boolean excludeOffensive;

private final boolean excludeProtected;

private UserFlagsExcludeFilter(

UserTable userTable,

boolean excludeAntisocial,

boolean excludeOffensive,

boolean excludeProtected) {

this.userTable = userTable;

this.excludeAntisocial = excludeAntisocial;

this.excludeOffensive = excludeOffensive;

this.excludeProtected = excludeProtected;

}

@Override

public int hashCode() {

return (excludeAntisocial ? 13 : 0) + (excludeOffensive ? 1 : 0) + (excludeProtected ? 2 : 0);

}

@Override

public boolean equals(Object obj) {

if (!(obj instanceof UserFlagsExcludeFilter)) {

return false;

}

UserFlagsExcludeFilter filter = UserFlagsExcludeFilter.class.cast(obj);

return (excludeAntisocial == filter.excludeAntisocial)

&& (excludeOffensive == filter.excludeOffensive)

&& (excludeProtected == filter.excludeProtected);

}

@Override

public String toString(String field) {

return "UserFlagsExcludeFilter";

}

@Override

public Weight createWeight(IndexSearcher searcher, ScoreMode scoreMode, float boost) {

return new DefaultFilterWeight(this) {

@Override

protected DocIdSetIterator getDocIdSetIterator(LeafReaderContext context) throws IOException {

LeafReader reader = context.reader();

if (userTable == null) {

return new AllDocsIterator(reader);

}

final int bits =

(excludeAntisocial ? UserTable.ANTISOCIAL\_BIT : 0)

| (excludeOffensive ? UserTable.OFFENSIVE\_BIT | UserTable.NSFW\_BIT : 0)

| (excludeProtected ? UserTable.IS\_PROTECTED\_BIT : 0);

if (bits != 0) {

return new UserFlagsExcludeDocIdSetIterator(reader, userTable) {

@Override

protected boolean checkUserFlags(UserTable table, long userID) {

return !table.isSet(userID, bits);

}

};

}

return new AllDocsIterator(reader);

}

};

}

private abstract static class UserFlagsExcludeDocIdSetIterator extends RangeFilterDISI {

private final UserTable userTable;

private final NumericDocValues fromUserID;

public UserFlagsExcludeDocIdSetIterator(

LeafReader indexReader, UserTable table) throws IOException {

super(indexReader);

userTable = table;

fromUserID =

indexReader.getNumericDocValues(EarlybirdFieldConstant.FROM\_USER\_ID\_CSF.getFieldName());

}

@Override

protected boolean shouldReturnDoc() throws IOException {

return fromUserID.advanceExact(docID())

&& checkUserFlags(userTable, fromUserID.longValue());

}

protected abstract boolean checkUserFlags(UserTable table, long userID);

}

}