package com.twitter.search.earlybird.queryparser;

import com.twitter.search.common.constants.QueryCacheConstants;

import com.twitter.search.queryparser.query.Conjunction;

import com.twitter.search.queryparser.query.Disjunction;

import com.twitter.search.queryparser.query.Phrase;

import com.twitter.search.queryparser.query.QueryParserException;

import com.twitter.search.queryparser.query.SpecialTerm;

import com.twitter.search.queryparser.query.Term;

import com.twitter.search.queryparser.query.search.SearchOperator;

import com.twitter.search.queryparser.query.search.SearchOperatorConstants;

import com.twitter.search.queryparser.query.search.SearchQueryVisitor;

/\*\*

\* Visitor to detect presence of any antisocial / spam operator in a Query.

\* Visitor returns true if any operators it detects were found.

\*/

public class DetectAntisocialVisitor extends SearchQueryVisitor<Boolean> {

// True if the query contains any operator to include antisocial tweets.

private boolean includeAntisocial = false;

// True if the query contains any operator to exclude antisocial/spam tweets.

private boolean excludeAntisocial = false;

// True if the query contains an antisocial tweets filter.

private boolean filterAntisocial = false;

public boolean hasIncludeAntisocial() {

return includeAntisocial;

}

public boolean hasExcludeAntisocial() {

return excludeAntisocial;

}

public boolean hasFilterAntisocial() {

return filterAntisocial;

}

public boolean hasAnyAntisocialOperator() {

// Top tweets is considered an antisocial operator due to scoring also excluding

// spam tweets.

return hasIncludeAntisocial() || hasExcludeAntisocial() || hasFilterAntisocial();

}

@Override public Boolean visit(Disjunction disjunction) throws QueryParserException {

boolean found = false;

for (com.twitter.search.queryparser.query.Query node : disjunction.getChildren()) {

if (node.accept(this)) {

found = true;

}

}

return found;

}

@Override public Boolean visit(Conjunction conjunction) throws QueryParserException {

boolean found = false;

for (com.twitter.search.queryparser.query.Query node : conjunction.getChildren()) {

if (node.accept(this)) {

found = true;

}

}

return found;

}

@Override public Boolean visit(SearchOperator operator) throws QueryParserException {

boolean found = false;

switch (operator.getOperatorType()) {

case INCLUDE:

if (SearchOperatorConstants.ANTISOCIAL.equals(operator.getOperand())) {

if (operator.mustNotOccur()) {

excludeAntisocial = true;

} else {

includeAntisocial = true;

}

found = true;

}

break;

case EXCLUDE:

if (SearchOperatorConstants.ANTISOCIAL.equals(operator.getOperand())) {

if (operator.mustNotOccur()) {

includeAntisocial = true;

} else {

excludeAntisocial = true;

}

found = true;

}

break;

case FILTER:

if (SearchOperatorConstants.ANTISOCIAL.equals(operator.getOperand())) {

if (operator.mustNotOccur()) {

excludeAntisocial = true;

} else {

filterAntisocial = true;

}

found = true;

}

break;

case CACHED\_FILTER:

if (QueryCacheConstants.EXCLUDE\_SPAM.equals(operator.getOperand())

|| QueryCacheConstants.EXCLUDE\_SPAM\_AND\_NATIVERETWEETS.equals(operator.getOperand())

|| QueryCacheConstants.EXCLUDE\_ANTISOCIAL.equals(operator.getOperand())

|| QueryCacheConstants.EXCLUDE\_ANTISOCIAL\_AND\_NATIVERETWEETS

.equals(operator.getOperand())) {

excludeAntisocial = true;

found = true;

}

break;

default:

break;

}

return found;

}

@Override

public Boolean visit(SpecialTerm special) throws QueryParserException {

return false;

}

@Override

public Boolean visit(Phrase phrase) throws QueryParserException {

return false;

}

@Override

public Boolean visit(Term term) throws QueryParserException {

return false;

}

}