package com.twitter.search.earlybird.querycache;

import java.util.Arrays;

import java.util.List;

import java.util.Set;

import com.google.common.collect.ImmutableList;

import com.google.common.collect.Sets;

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

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

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

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

import static com.twitter.search.common.util.RuleBasedConverter.Rule;

/\*\*

\* Rules to convert exclude operators into cached filters and consolidate them.

\* NOTE: this is copied from blender/core/parser/service/queryparser/QueryCacheConversionRules.java

\* We should remove the blender one once this is in production.

\*/

public final class QueryCacheConversionRules {

static final SearchOperator EXCLUDE\_ANTISOCIAL =

new SearchOperator(SearchOperator.Type.EXCLUDE, SearchOperatorConstants.ANTISOCIAL);

static final SearchOperator EXCLUDE\_SPAM =

new SearchOperator(SearchOperator.Type.EXCLUDE, SearchOperatorConstants.SPAM);

static final SearchOperator EXCLUDE\_REPLIES =

new SearchOperator(SearchOperator.Type.EXCLUDE, SearchOperatorConstants.REPLIES);

static final SearchOperator EXCLUDE\_NATIVERETWEETS =

new SearchOperator(SearchOperator.Type.EXCLUDE, SearchOperatorConstants.NATIVE\_RETWEETS);

public static final SearchOperator CACHED\_EXCLUDE\_ANTISOCIAL =

new SearchOperator(SearchOperator.Type.CACHED\_FILTER,

QueryCacheConstants.EXCLUDE\_ANTISOCIAL);

static final SearchOperator CACHED\_EXCLUDE\_NATIVERETWEETS =

new SearchOperator(SearchOperator.Type.CACHED\_FILTER,

QueryCacheConstants.EXCLUDE\_ANTISOCIAL\_AND\_NATIVERETWEETS);

static final SearchOperator CACHED\_EXCLUDE\_SPAM =

new SearchOperator(SearchOperator.Type.CACHED\_FILTER,

QueryCacheConstants.EXCLUDE\_SPAM);

static final SearchOperator CACHED\_EXCLUDE\_SPAM\_AND\_NATIVERETWEETS =

new SearchOperator(SearchOperator.Type.CACHED\_FILTER,

QueryCacheConstants.EXCLUDE\_SPAM\_AND\_NATIVERETWEETS);

static final SearchOperator CACHED\_EXCLUDE\_REPLIES =

new SearchOperator(SearchOperator.Type.CACHED\_FILTER,

QueryCacheConstants.EXCLUDE\_REPLIES);

private QueryCacheConversionRules() {

}

public static final List<Rule<Query>> DEFAULT\_RULES = ImmutableList.of(

// basic translation from exclude:filter to cached filter

new Rule<>(new Query[]{EXCLUDE\_ANTISOCIAL},

new Query[]{CACHED\_EXCLUDE\_ANTISOCIAL}),

new Rule<>(new Query[]{EXCLUDE\_SPAM},

new Query[]{CACHED\_EXCLUDE\_SPAM}),

new Rule<>(new Query[]{EXCLUDE\_NATIVERETWEETS},

new Query[]{CACHED\_EXCLUDE\_NATIVERETWEETS}),

new Rule<>(new Query[]{EXCLUDE\_REPLIES},

new Query[]{CACHED\_EXCLUDE\_REPLIES}),

// combine two cached filter to a new one

new Rule<>(new Query[]{CACHED\_EXCLUDE\_SPAM, CACHED\_EXCLUDE\_NATIVERETWEETS},

new Query[]{CACHED\_EXCLUDE\_SPAM\_AND\_NATIVERETWEETS}),

// Remove redundant filters. A cached filter is redundant when it coexist with a

// more strict filter. Note all the filter will filter out antisocial.

new Rule<>(

new Query[]{CACHED\_EXCLUDE\_SPAM, CACHED\_EXCLUDE\_ANTISOCIAL},

new Query[]{CACHED\_EXCLUDE\_SPAM}),

new Rule<>(

new Query[]{CACHED\_EXCLUDE\_NATIVERETWEETS, CACHED\_EXCLUDE\_ANTISOCIAL},

new Query[]{CACHED\_EXCLUDE\_NATIVERETWEETS}),

new Rule<>(

new Query[]{CACHED\_EXCLUDE\_SPAM\_AND\_NATIVERETWEETS, CACHED\_EXCLUDE\_ANTISOCIAL},

new Query[]{CACHED\_EXCLUDE\_SPAM\_AND\_NATIVERETWEETS}),

new Rule<>(

new Query[]{CACHED\_EXCLUDE\_SPAM\_AND\_NATIVERETWEETS, CACHED\_EXCLUDE\_SPAM},

new Query[]{CACHED\_EXCLUDE\_SPAM\_AND\_NATIVERETWEETS}),

new Rule<>(

new Query[]{CACHED\_EXCLUDE\_SPAM\_AND\_NATIVERETWEETS, CACHED\_EXCLUDE\_NATIVERETWEETS},

new Query[]{CACHED\_EXCLUDE\_SPAM\_AND\_NATIVERETWEETS})

);

public static final List<Query> STRIP\_ANNOTATIONS\_QUERIES;

static {

Set<Query> stripAnnotationsQueries = Sets.newHashSet();

for (Rule<Query> rule : DEFAULT\_RULES) {

stripAnnotationsQueries.addAll(Arrays.asList(rule.getSources()));

}

STRIP\_ANNOTATIONS\_QUERIES = ImmutableList.copyOf(stripAnnotationsQueries);

}

}