package com.twitter.search.earlybird\_root.filters;

import java.util.Collections;

import java.util.List;

import java.util.Map;

import javax.annotation.Nullable;

import com.google.common.annotations.VisibleForTesting;

import com.google.common.collect.Lists;

import com.google.common.collect.Maps;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.search.common.decider.SearchDecider;

import com.twitter.search.common.metrics.SearchCounter;

import com.twitter.search.earlybird.config.ServingRange;

import com.twitter.search.earlybird\_root.common.EarlybirdRequestContext;

import com.twitter.search.earlybird\_root.common.EarlybirdRequestType;

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

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

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

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

/\*\*

\* Adds query filters that filter out tweets outside a tier's serving range. Two tiers might load

\* the same timeslice, so if the filtering is not done, the two tiers might return duplicates. The

\* mergers should know how to handle the duplicates, but this might decrease the number or the

\* quality of the returned results.

\*/

public class EarlybirdTimeFilterQueryRewriter {

private static final Logger LOG =

LoggerFactory.getLogger(EarlybirdTimeFilterQueryRewriter.class);

private static final Map<EarlybirdRequestType, SearchCounter> NO\_QUERY\_COUNTS;

static {

final Map<EarlybirdRequestType, SearchCounter> tempMap =

Maps.newEnumMap(EarlybirdRequestType.class);

for (EarlybirdRequestType requestType : EarlybirdRequestType.values()) {

tempMap.put(requestType, SearchCounter.export(

"time\_filter\_query\_rewriter\_" + requestType.getNormalizedName() + "\_no\_query\_count"));

}

NO\_QUERY\_COUNTS = Collections.unmodifiableMap(tempMap);

}

@VisibleForTesting

static final Map<EarlybirdRequestType, String> ADD\_SINCE\_ID\_MAX\_ID\_DECIDER\_KEY\_MAP;

static {

final String ADD\_SINCE\_ID\_MAX\_ID\_DECIDER\_KEY\_TEMPLATE =

"add\_since\_id\_max\_id\_operators\_to\_%s\_query";

final Map<EarlybirdRequestType, String> tempMap = Maps.newEnumMap(EarlybirdRequestType.class);

for (EarlybirdRequestType requestType : EarlybirdRequestType.values()) {

tempMap.put(

requestType,

String.format(ADD\_SINCE\_ID\_MAX\_ID\_DECIDER\_KEY\_TEMPLATE, requestType.getNormalizedName()));

}

ADD\_SINCE\_ID\_MAX\_ID\_DECIDER\_KEY\_MAP = Collections.unmodifiableMap(tempMap);

}

@VisibleForTesting

static final String ADD\_SINCE\_ID\_MAX\_ID\_TO\_NULL\_SERIALIZED\_QUERIES\_DECIDER\_KEY =

"add\_since\_id\_max\_id\_operators\_to\_null\_serialized\_queries";

private final SearchDecider decider;

private final ServingRangeProvider servingRangeProvider;

EarlybirdTimeFilterQueryRewriter(

ServingRangeProvider servingRangeProvider,

SearchDecider decider) {

this.servingRangeProvider = servingRangeProvider;

this.decider = decider;

}

/\*\*

\* Add maxId and sinceId fields to the serialized query.

\*

\* This must be done after calculating the IdTimeRanges to prevent interfering with calculating

\* IdTimeRanges

\*/

public EarlybirdRequestContext rewriteRequest(EarlybirdRequestContext requestContext)

throws QueryParserException {

Query q = requestContext.getParsedQuery();

if (q == null) {

if (requestContext.getEarlybirdRequestType() != EarlybirdRequestType.TERM\_STATS) {

LOG.warn("Received request without a parsed query: " + requestContext.getRequest());

NO\_QUERY\_COUNTS.get(requestContext.getEarlybirdRequestType()).increment();

}

if (!decider.isAvailable(ADD\_SINCE\_ID\_MAX\_ID\_TO\_NULL\_SERIALIZED\_QUERIES\_DECIDER\_KEY)) {

return requestContext;

}

}

return addOperators(requestContext, q);

}

private EarlybirdRequestContext addOperators(

EarlybirdRequestContext requestContext,

@Nullable Query query) throws QueryParserException {

// Add the SINCE\_ID and MAX\_ID operators only if the decider is enabled.

if (!decider.isAvailable(

ADD\_SINCE\_ID\_MAX\_ID\_DECIDER\_KEY\_MAP.get(requestContext.getEarlybirdRequestType()))) {

return requestContext;

}

// Note: can't recompute the search operators because the serving range changes in real time

// for the most recent tier.

ServingRange servingRange = servingRangeProvider.getServingRange(

requestContext, requestContext.useOverrideTierConfig());

long tierSinceId = servingRange.getServingRangeSinceId();

SearchOperator sinceId = new SearchOperator(SearchOperator.Type.SINCE\_ID,

Long.toString(tierSinceId));

long tierMaxId = servingRange.getServingRangeMaxId();

SearchOperator maxId = new SearchOperator(SearchOperator.Type.MAX\_ID,

Long.toString(tierMaxId));

List<Query> conjunctionChildren = (query == null)

? Lists.<Query>newArrayList(sinceId, maxId)

: Lists.newArrayList(query, sinceId, maxId);

Query restrictedQuery = new Conjunction(conjunctionChildren).simplify();

EarlybirdRequestContext copiedRequestContext =

EarlybirdRequestContext.copyRequestContext(requestContext, restrictedQuery);

return copiedRequestContext;

}

}