package com.twitter.cr\_mixer.util

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

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

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

import com.twitter.search.queryparser.query.{Query => EbQuery}

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

import scala.collection.JavaConverters.\_

import com.twitter.search.earlybird.thriftscala.ThriftSearchResultMetadataOptions

import com.twitter.simclusters\_v2.common.TweetId

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

import com.twitter.util.Duration

import com.twitter.search.common.query.thriftjava.thriftscala.CollectorTerminationParams

object EarlybirdSearchUtil {

val EarlybirdClientId: String = "cr-mixer.prod"

val Mentions: String = EarlybirdFieldConstant.MENTIONS\_FACET

val Hashtags: String = EarlybirdFieldConstant.HASHTAGS\_FACET

val FacetsToFetch: Seq[String] = Seq(Mentions, Hashtags)

val MetadataOptions: ThriftSearchResultMetadataOptions = ThriftSearchResultMetadataOptions(

getTweetUrls = true,

getResultLocation = false,

getLuceneScore = false,

getInReplyToStatusId = true,

getReferencedTweetAuthorId = true,

getMediaBits = true,

getAllFeatures = true,

getFromUserId = true,

returnSearchResultFeatures = true,

// Set getExclusiveConversationAuthorId in order to retrieve Exclusive / SuperFollow tweets.

getExclusiveConversationAuthorId = true

)

// Filter out retweets and replies

val TweetTypesToExclude: Seq[String] =

Seq(

SearchOperatorConstants.NATIVE\_RETWEETS,

SearchOperatorConstants.REPLIES)

def GetCollectorTerminationParams(

maxNumHitsPerShard: Int,

processingTimeout: Duration

): Option[CollectorTerminationParams] = {

Some(

CollectorTerminationParams(

// maxHitsToProcess is used for early termination on each EB shard

maxHitsToProcess = Some(maxNumHitsPerShard),

timeoutMs = processingTimeout.inMilliseconds.toInt

))

}

/\*\*

\* Get EarlybirdQuery

\* This function creates a EBQuery based on the search input

\*/

def GetEarlybirdQuery(

beforeTweetIdExclusive: Option[TweetId],

afterTweetIdExclusive: Option[TweetId],

excludedTweetIds: Set[TweetId],

filterOutRetweetsAndReplies: Boolean

): Option[EbQuery] =

CreateConjunction(

Seq(

CreateRangeQuery(beforeTweetIdExclusive, afterTweetIdExclusive),

CreateExcludedTweetIdsQuery(excludedTweetIds),

CreateTweetTypesFilters(filterOutRetweetsAndReplies)

).flatten)

def CreateRangeQuery(

beforeTweetIdExclusive: Option[TweetId],

afterTweetIdExclusive: Option[TweetId]

): Option[EbQuery] = {

val beforeIdClause = beforeTweetIdExclusive.map { beforeId =>

// MAX\_ID is an inclusive value therefore we subtract 1 from beforeId.

new SearchOperator(SearchOperator.Type.MAX\_ID, (beforeId - 1).toString)

}

val afterIdClause = afterTweetIdExclusive.map { afterId =>

new SearchOperator(SearchOperator.Type.SINCE\_ID, afterId.toString)

}

CreateConjunction(Seq(beforeIdClause, afterIdClause).flatten)

}

def CreateTweetTypesFilters(filterOutRetweetsAndReplies: Boolean): Option[EbQuery] = {

if (filterOutRetweetsAndReplies) {

val tweetTypeFilters = TweetTypesToExclude.map { searchOperator =>

new SearchOperator(SearchOperator.Type.EXCLUDE, searchOperator)

}

CreateConjunction(tweetTypeFilters)

} else None

}

def CreateConjunction(clauses: Seq[EbQuery]): Option[EbQuery] = {

clauses.size match {

case 0 => None

case 1 => Some(clauses.head)

case \_ => Some(new Conjunction(clauses.asJava))

}

}

def CreateExcludedTweetIdsQuery(tweetIds: Set[TweetId]): Option[EbQuery] = {

if (tweetIds.nonEmpty) {

Some(

new SearchOperator.Builder()

.setType(SearchOperator.Type.NAMED\_MULTI\_TERM\_DISJUNCTION)

.addOperand(EarlybirdFieldConstant.ID\_FIELD.getFieldName)

.addOperand(EXCLUDE\_TWEET\_IDS)

.setOccur(Query.Occur.MUST\_NOT)

.build())

} else None

}

/\*\*

\* Get NamedDisjunctions with excludedTweetIds

\*/

def GetNamedDisjunctions(excludedTweetIds: Set[TweetId]): Option[Map[String, Seq[Long]]] =

if (excludedTweetIds.nonEmpty)

createNamedDisjunctionsExcludedTweetIds(excludedTweetIds)

else None

val EXCLUDE\_TWEET\_IDS = "exclude\_tweet\_ids"

private def createNamedDisjunctionsExcludedTweetIds(

tweetIds: Set[TweetId]

): Option[Map[String, Seq[Long]]] = {

if (tweetIds.nonEmpty) {

Some(Map(EXCLUDE\_TWEET\_IDS -> tweetIds.toSeq))

} else None

}

}