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

import com.google.common.base.Optional;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.search.common.partitioning.snowflakeparser.SnowflakeIdParser;

import com.twitter.search.earlybird.thrift.EarlybirdRequest;

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

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

import com.twitter.search.queryparser.util.IdTimeRanges;

public final class EarlybirdRequestUtil {

private static final Logger LOG = LoggerFactory.getLogger(EarlybirdRequestUtil.class);

private EarlybirdRequestUtil() {

}

/\*\*

\* Returns the max ID specified in the query. The max ID is determined based on the max\_id

\* operator, and the returned value is an inclusive max ID (that is, the returned response is

\* allowed to have a tweet with this ID).

\*

\* If the query is null, could not be parsed or does not have a max\_id operator, Optional.absent()

\* is returned.

\*

\* @param query The query.

\* @return The max ID specified in the given query (based on the max\_id operator).

\*/

public static Optional<Long> getRequestMaxId(Query query) {

if (query == null) {

return Optional.absent();

}

IdTimeRanges idTimeRanges = null;

try {

idTimeRanges = IdTimeRanges.fromQuery(query);

} catch (QueryParserException e) {

LOG.warn("Exception while getting max\_id/until\_time from query: " + query, e);

}

if (idTimeRanges == null) {

// An exception was thrown or the query doesn't accept the boundary operators.

return Optional.absent();

}

return idTimeRanges.getMaxIDInclusive();

}

/\*\*

\* Returns the max ID specified in the query, based on the until\_time operator. The returned ID

\* is inclusive (that is, the returned response is allowed to have a tweet with this ID).

\*

\* If the query is null, could not be parsed or does not have an until\_time operator,

\* Optional.absent() is returned.

\*

\* @param query The query.

\* @return The max ID specified in the given query (based on the until\_time operator).

\*/

public static Optional<Long> getRequestMaxIdFromUntilTime(Query query) {

if (query == null) {

return Optional.absent();

}

IdTimeRanges idTimeRanges = null;

try {

idTimeRanges = IdTimeRanges.fromQuery(query);

} catch (QueryParserException e) {

LOG.warn("Exception while getting max\_id/until\_time from query: " + query, e);

}

if (idTimeRanges == null) {

// An exception was thrown or the query doesn't accept the boundary operators.

return Optional.absent();

}

Optional<Integer> queryUntilTimeExclusive = idTimeRanges.getUntilTimeExclusive();

Optional<Long> maxId = Optional.absent();

if (queryUntilTimeExclusive.isPresent()) {

long timestampMillis = queryUntilTimeExclusive.get() \* 1000L;

if (SnowflakeIdParser.isUsableSnowflakeTimestamp(timestampMillis)) {

// Convert timestampMillis to an ID, and subtract 1, because the until\_time operator is

// exclusive, and we need to return an inclusive max ID.

maxId = Optional.of(SnowflakeIdParser.generateValidStatusId(timestampMillis, 0) - 1);

}

}

return maxId;

}

/\*\*

\* Creates a copy of the given EarlybirdRequest and unsets all fields that are used

\* only by the SuperRoot.

\*/

public static EarlybirdRequest unsetSuperRootFields(

EarlybirdRequest request, boolean unsetFollowedUserIds) {

EarlybirdRequest newRequest = request.deepCopy();

newRequest.unsetGetOlderResults();

newRequest.unsetGetProtectedTweetsOnly();

if (unsetFollowedUserIds) {

newRequest.unsetFollowedUserIds();

}

newRequest.unsetAdjustedProtectedRequestParams();

newRequest.unsetAdjustedFullArchiveRequestParams();

return newRequest;

}

}