package com.twitter.tweetypie.storage

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.stitch.Stitch

import com.twitter.storage.client.manhattan.kv.DeniedManhattanException

import com.twitter.tweetypie.storage.TweetUtils.\_

import com.twitter.util.Throw

import com.twitter.util.Time

object DeleteAdditionalFieldsHandler {

def apply(

delete: ManhattanOperations.Delete,

stats: StatsReceiver

): TweetStorageClient.DeleteAdditionalFields =

(unfilteredTweetIds: Seq[TweetId], additionalFields: Seq[Field]) => {

val tweetIds = unfilteredTweetIds.filter(\_ > 0)

val additionalFieldIds = additionalFields.map(\_.id)

require(additionalFields.nonEmpty, "Additional fields to delete cannot be empty")

require(

additionalFieldIds.min >= TweetFields.firstAdditionalFieldId,

s"Additional fields $additionalFields must be in additional field range (>= ${TweetFields.firstAdditionalFieldId})"

)

Stats.addWidthStat("deleteAdditionalFields", "tweetIds", tweetIds.size, stats)

Stats.addWidthStat(

"deleteAdditionalFields",

"additionalFieldIds",

additionalFieldIds.size,

stats

)

Stats.updatePerFieldQpsCounters(

"deleteAdditionalFields",

additionalFieldIds,

tweetIds.size,

stats

)

val mhTimestamp = Time.now

val stitches = tweetIds.map { tweetId =>

val (fieldIds, mhKeysToDelete) =

additionalFieldIds.map { fieldId =>

(fieldId, TweetKey.additionalFieldsKey(tweetId, fieldId))

}.unzip

val deletionStitches = mhKeysToDelete.map { mhKeyToDelete =>

delete(mhKeyToDelete, Some(mhTimestamp)).liftToTry

}

Stitch.collect(deletionStitches).map { responsesTries =>

val wasRateLimited = responsesTries.exists {

case Throw(e: DeniedManhattanException) => true

case \_ => false

}

val resultsPerTweet = fieldIds.zip(responsesTries).toMap

if (wasRateLimited) {

buildTweetOverCapacityResponse("deleteAdditionalFields", tweetId, resultsPerTweet)

} else {

buildTweetResponse("deleteAdditionalFields", tweetId, resultsPerTweet)

}

}

}

Stitch.collect(stitches)

}

}