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.Response.TweetResponseCode

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

import com.twitter.tweetypie.storage\_internal.thriftscala.StoredTweet

import com.twitter.tweetypie.thriftscala.DeletedTweet

import scala.util.control.NonFatal

sealed trait DeleteState

object DeleteState {

/\*\*

\* This tweet is deleted but has not been permanently deleted from Manhattan. Tweets in this state

\* may be undeleted.

\*/

case object SoftDeleted extends DeleteState

/\*\*

\* This tweet is deleted after being bounced for violating the Twitter Rules but has not been

\* permanently deleted from Manhattan. Tweets in this state may NOT be undeleted.

\*/

case object BounceDeleted extends DeleteState

/\*\*

\* This tweet has been permanently deleted from Manhattan.

\*/

case object HardDeleted extends DeleteState

/\*\*

\* There is no data in Manhattan to distinguish this tweet id from one that never existed.

\*/

case object NotFound extends DeleteState

/\*\*

\* This tweet exists and is not in a deleted state.

\*/

case object NotDeleted extends DeleteState

}

case class DeletedTweetResponse(

tweetId: TweetId,

overallResponse: TweetResponseCode,

deleteState: DeleteState,

tweet: Option[DeletedTweet])

object GetDeletedTweetsHandler {

def apply(

read: ManhattanOperations.Read,

stats: StatsReceiver

): TweetStorageClient.GetDeletedTweets =

(unfilteredTweetIds: Seq[TweetId]) => {

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

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

val stitches = tweetIds.map { tweetId =>

read(tweetId)

.map { mhRecords =>

val storedTweet = buildStoredTweet(tweetId, mhRecords)

TweetStateRecord.mostRecent(mhRecords) match {

case Some(m: TweetStateRecord.SoftDeleted) => softDeleted(m, storedTweet)

case Some(m: TweetStateRecord.BounceDeleted) => bounceDeleted(m, storedTweet)

case Some(m: TweetStateRecord.HardDeleted) => hardDeleted(m, storedTweet)

case \_ if storedTweet.getFieldBlobs(expectedFields).isEmpty => notFound(tweetId)

case \_ => notDeleted(tweetId, storedTweet)

}

}

.handle {

case \_: DeniedManhattanException =>

DeletedTweetResponse(

tweetId,

TweetResponseCode.OverCapacity,

DeleteState.NotFound,

None

)

case NonFatal(ex) =>

TweetUtils.log.warning(

ex,

s"Unhandled exception in GetDeletedTweetsHandler for tweetId: $tweetId"

)

DeletedTweetResponse(tweetId, TweetResponseCode.Failure, DeleteState.NotFound, None)

}

}

Stitch.collect(stitches)

}

private def notFound(tweetId: TweetId) =

DeletedTweetResponse(

tweetId = tweetId,

overallResponse = TweetResponseCode.Success,

deleteState = DeleteState.NotFound,

tweet = None

)

private def softDeleted(record: TweetStateRecord.SoftDeleted, storedTweet: StoredTweet) =

DeletedTweetResponse(

record.tweetId,

TweetResponseCode.Success,

DeleteState.SoftDeleted,

Some(

StorageConversions

.toDeletedTweet(storedTweet)

.copy(deletedAtMsec = Some(record.createdAt))

)

)

private def bounceDeleted(record: TweetStateRecord.BounceDeleted, storedTweet: StoredTweet) =

DeletedTweetResponse(

record.tweetId,

TweetResponseCode.Success,

DeleteState.BounceDeleted,

Some(

StorageConversions

.toDeletedTweet(storedTweet)

.copy(deletedAtMsec = Some(record.createdAt))

)

)

private def hardDeleted(record: TweetStateRecord.HardDeleted, storedTweet: StoredTweet) =

DeletedTweetResponse(

record.tweetId,

TweetResponseCode.Success,

DeleteState.HardDeleted,

Some(

StorageConversions

.toDeletedTweet(storedTweet)

.copy(

hardDeletedAtMsec = Some(record.createdAt),

deletedAtMsec = Some(record.deletedAt)

)

)

)

/\*\*

\* notDeleted returns a tweet to simplify tweetypie.handler.UndeleteTweetHandler

\*/

private def notDeleted(tweetId: TweetId, storedTweet: StoredTweet) =

DeletedTweetResponse(

tweetId = tweetId,

overallResponse = TweetResponseCode.Success,

deleteState = DeleteState.NotDeleted,

tweet = Some(StorageConversions.toDeletedTweet(storedTweet))

)

}