package com.twitter.tweetypie.storage

import com.twitter.mediaservices.commons.tweetmedia.thriftscala.\_

import com.twitter.scrooge.TFieldBlob

import com.twitter.tweetypie.additionalfields.AdditionalFields

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

import com.twitter.tweetypie.thriftscala.\_

import com.twitter.tweetypie.util.TweetLenses

object StorageConversions {

private val tbTweetCompiledAdditionalFieldIds =

StoredTweet.metaData.fields.map(\_.id).filter(AdditionalFields.isAdditionalFieldId)

def toStoredReply(reply: Reply, conversationId: Option[TweetId]): StoredReply =

StoredReply(

inReplyToStatusId = reply.inReplyToStatusId.getOrElse(0),

inReplyToUserId = reply.inReplyToUserId,

conversationId = conversationId

)

def toStoredShare(share: Share): StoredShare =

StoredShare(

share.sourceStatusId,

share.sourceUserId,

share.parentStatusId

)

def toStoredQuotedTweet(qt: QuotedTweet, text: String): Option[StoredQuotedTweet] =

qt.permalink

.filterNot { p =>

text.contains(p.shortUrl)

} // omit StoredQuotedTweet when url already in text

.map { p =>

StoredQuotedTweet(

qt.tweetId,

qt.userId,

p.shortUrl

)

}

def toStoredGeo(tweet: Tweet): Option[StoredGeo] =

TweetLenses.geoCoordinates.get(tweet) match {

case None =>

TweetLenses.placeId.get(tweet) match {

case None => None

case Some(placeId) =>

Some(

StoredGeo(

latitude = 0.0,

longitude = 0.0,

geoPrecision = 0,

entityId = 0,

name = Some(placeId)

)

)

}

case Some(coords) =>

Some(

StoredGeo(

latitude = coords.latitude,

longitude = coords.longitude,

geoPrecision = coords.geoPrecision,

entityId = if (coords.display) 2 else 0,

name = TweetLenses.placeId.get(tweet)

)

)

}

def toStoredMedia(mediaList: Seq[MediaEntity]): Seq[StoredMediaEntity] =

mediaList.filter(\_.sourceStatusId.isEmpty).flatMap(toStoredMediaEntity)

def toStoredMediaEntity(media: MediaEntity): Option[StoredMediaEntity] =

media.sizes.find(\_.sizeType == MediaSizeType.Orig).map { origSize =>

StoredMediaEntity(

id = media.mediaId,

mediaType = origSize.deprecatedContentType.value.toByte,

width = origSize.width.toShort,

height = origSize.height.toShort

)

}

// The language and ids fields are for compatibility with existing tweets stored in manhattan.

def toStoredNarrowcast(narrowcast: Narrowcast): StoredNarrowcast =

StoredNarrowcast(

language = Some(Seq.empty),

location = Some(narrowcast.location),

ids = Some(Seq.empty)

)

def toStoredAdditionalFields(from: Seq[TFieldBlob], to: StoredTweet): StoredTweet =

from.foldLeft(to) { case (t, f) => t.setField(f) }

def toStoredAdditionalFields(from: Tweet, to: StoredTweet): StoredTweet =

toStoredAdditionalFields(AdditionalFields.additionalFields(from), to)

def toStoredTweet(tweet: Tweet): StoredTweet = {

val storedTweet =

StoredTweet(

id = tweet.id,

userId = Some(TweetLenses.userId(tweet)),

text = Some(TweetLenses.text(tweet)),

createdVia = Some(TweetLenses.createdVia(tweet)),

createdAtSec = Some(TweetLenses.createdAt(tweet)),

reply =

TweetLenses.reply(tweet).map { r => toStoredReply(r, TweetLenses.conversationId(tweet)) },

share = TweetLenses.share(tweet).map(toStoredShare),

contributorId = tweet.contributor.map(\_.userId),

geo = toStoredGeo(tweet),

hasTakedown = Some(TweetLenses.hasTakedown(tweet)),

nsfwUser = Some(TweetLenses.nsfwUser(tweet)),

nsfwAdmin = Some(TweetLenses.nsfwAdmin(tweet)),

media = tweet.media.map(toStoredMedia),

narrowcast = TweetLenses.narrowcast(tweet).map(toStoredNarrowcast),

nullcast = Some(TweetLenses.nullcast(tweet)),

trackingId = TweetLenses.trackingId(tweet),

quotedTweet = TweetLenses.quotedTweet(tweet).flatMap { qt =>

toStoredQuotedTweet(qt, TweetLenses.text(tweet))

}

)

toStoredAdditionalFields(tweet, storedTweet)

}

/\*\*

\* Does not need core data to be set. Constructs on disk tweet by avoiding the TweetLenses object

\* and only extracting the specified fields.

\*

\* NOTE: Assumes that specified fields are set in the tweet.

\*

\* @param tpTweet Tweetypie Tweet to be converted

\* @param fields the fields to be populated in the on disk Tweet

\*

\* @return an on disk Tweet which has only the specified fields set

\*/

def toStoredTweetForFields(tpTweet: Tweet, fields: Set[Field]): StoredTweet = {

// Make sure all the passed in fields are known or additional fields

require(

(fields -- Field.AllUpdatableCompiledFields)

.forall(field => AdditionalFields.isAdditionalFieldId(field.id))

)

val storedTweet =

StoredTweet(

id = tpTweet.id,

geo = if (fields.contains(Field.Geo)) {

tpTweet.coreData.get.coordinates match {

case None =>

tpTweet.coreData.get.placeId match {

case None => None

case Some(placeId) =>

Some(

StoredGeo(

latitude = 0.0,

longitude = 0.0,

geoPrecision = 0,

entityId = 0,

name = Some(placeId)

)

)

}

case Some(coords) =>

Some(

StoredGeo(

latitude = coords.latitude,

longitude = coords.longitude,

geoPrecision = coords.geoPrecision,

entityId = if (coords.display) 2 else 0,

name = tpTweet.coreData.get.placeId

)

)

}

} else {

None

},

hasTakedown =

if (fields.contains(Field.HasTakedown))

Some(tpTweet.coreData.get.hasTakedown)

else

None,

nsfwUser =

if (fields.contains(Field.NsfwUser))

Some(tpTweet.coreData.get.nsfwUser)

else

None,

nsfwAdmin =

if (fields.contains(Field.NsfwAdmin))

Some(tpTweet.coreData.get.nsfwAdmin)

else

None

)

if (fields.map(\_.id).exists(AdditionalFields.isAdditionalFieldId))

toStoredAdditionalFields(tpTweet, storedTweet)

else

storedTweet

}

def fromStoredReply(reply: StoredReply): Reply =

Reply(

Some(reply.inReplyToStatusId).filter(\_ > 0),

reply.inReplyToUserId

)

def fromStoredShare(share: StoredShare): Share =

Share(

share.sourceStatusId,

share.sourceUserId,

share.parentStatusId

)

def fromStoredQuotedTweet(qt: StoredQuotedTweet): QuotedTweet =

QuotedTweet(

qt.tweetId,

qt.userId,

Some(

ShortenedUrl(

shortUrl = qt.shortUrl,

longUrl = "", // will be hydrated later via tweetypie's QuotedTweetRefUrlsHydrator

displayText = "" //will be hydrated later via tweetypie's QuotedTweetRefUrlsHydrator

)

)

)

def fromStoredGeo(geo: StoredGeo): GeoCoordinates =

GeoCoordinates(

latitude = geo.latitude,

longitude = geo.longitude,

geoPrecision = geo.geoPrecision,

display = geo.entityId == 2

)

def fromStoredMediaEntity(media: StoredMediaEntity): MediaEntity =

MediaEntity(

fromIndex = -1, // will get filled in later

toIndex = -1, // will get filled in later

url = null, // will get filled in later

mediaPath = "", // field is obsolete

mediaUrl = null, // will get filled in later

mediaUrlHttps = null, // will get filled in later

displayUrl = null, // will get filled in later

expandedUrl = null, // will get filled in later

mediaId = media.id,

nsfw = false,

sizes = Set(

MediaSize(

sizeType = MediaSizeType.Orig,

resizeMethod = MediaResizeMethod.Fit,

deprecatedContentType = MediaContentType(media.mediaType),

width = media.width,

height = media.height

)

)

)

def fromStoredNarrowcast(narrowcast: StoredNarrowcast): Narrowcast =

Narrowcast(

location = narrowcast.location.getOrElse(Seq())

)

def fromStoredTweet(storedTweet: StoredTweet): Tweet = {

val coreData =

TweetCoreData(

userId = storedTweet.userId.get,

text = storedTweet.text.get,

createdVia = storedTweet.createdVia.get,

createdAtSecs = storedTweet.createdAtSec.get,

reply = storedTweet.reply.map(fromStoredReply),

share = storedTweet.share.map(fromStoredShare),

hasTakedown = storedTweet.hasTakedown.getOrElse(false),

nsfwUser = storedTweet.nsfwUser.getOrElse(false),

nsfwAdmin = storedTweet.nsfwAdmin.getOrElse(false),

narrowcast = storedTweet.narrowcast.map(fromStoredNarrowcast),

nullcast = storedTweet.nullcast.getOrElse(false),

trackingId = storedTweet.trackingId,

conversationId = storedTweet.reply.flatMap(\_.conversationId),

placeId = storedTweet.geo.flatMap(\_.name),

coordinates = storedTweet.geo.map(fromStoredGeo),

hasMedia = if (storedTweet.media.exists(\_.nonEmpty)) Some(true) else None

)

// retweets should never have their media, but some tweets incorrectly do.

val storedMedia = if (coreData.share.isDefined) Nil else storedTweet.media.toSeq

val tpTweet =

Tweet(

id = storedTweet.id,

coreData = Some(coreData),

contributor = storedTweet.contributorId.map(Contributor(\_)),

media = Some(storedMedia.flatten.map(fromStoredMediaEntity)),

mentions = Some(Seq.empty),

urls = Some(Seq.empty),

cashtags = Some(Seq.empty),

hashtags = Some(Seq.empty),

quotedTweet = storedTweet.quotedTweet.map(fromStoredQuotedTweet)

)

fromStoredAdditionalFields(storedTweet, tpTweet)

}

def fromStoredTweetAllowInvalid(storedTweet: StoredTweet): Tweet = {

fromStoredTweet(

storedTweet.copy(

userId = storedTweet.userId.orElse(Some(-1L)),

text = storedTweet.text.orElse(Some("")),

createdVia = storedTweet.createdVia.orElse(Some("")),

createdAtSec = storedTweet.createdAtSec.orElse(Some(-1L))

))

}

def fromStoredAdditionalFields(from: StoredTweet, to: Tweet): Tweet = {

val passThroughAdditionalFields =

from.\_passthroughFields.filterKeys(AdditionalFields.isAdditionalFieldId)

val allAdditionalFields =

from.getFieldBlobs(tbTweetCompiledAdditionalFieldIds) ++ passThroughAdditionalFields

allAdditionalFields.values.foldLeft(to) { case (t, f) => t.setField(f) }

}

def toDeletedTweet(storedTweet: StoredTweet): DeletedTweet = {

val noteTweetBlob = storedTweet.getFieldBlob(Tweet.NoteTweetField.id)

val noteTweetOption = noteTweetBlob.map(blob => NoteTweet.decode(blob.read))

DeletedTweet(

id = storedTweet.id,

userId = storedTweet.userId,

text = storedTweet.text,

createdAtSecs = storedTweet.createdAtSec,

share = storedTweet.share.map(toDeletedShare),

media = storedTweet.media.map(\_.map(toDeletedMediaEntity)),

noteTweetId = noteTweetOption.map(\_.id),

isExpandable = noteTweetOption.flatMap(\_.isExpandable)

)

}

def toDeletedShare(storedShare: StoredShare): DeletedTweetShare =

DeletedTweetShare(

sourceStatusId = storedShare.sourceStatusId,

sourceUserId = storedShare.sourceUserId,

parentStatusId = storedShare.parentStatusId

)

def toDeletedMediaEntity(storedMediaEntity: StoredMediaEntity): DeletedTweetMediaEntity =

DeletedTweetMediaEntity(

id = storedMediaEntity.id,

mediaType = storedMediaEntity.mediaType,

width = storedMediaEntity.width,

height = storedMediaEntity.height

)

}