package com.twitter.tweetypie.storage

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

import com.twitter.util.Time

/\*\*

\* A [[TweetStateRecord]] represents an action taken on a tweet and can be used to determine a tweet's state.

\*

\* The state is determined by the record with the most recent timestamp. In the absence of any

\* record a tweet is considered found, which is to say the tweet has not been through the

\* deletion process.

\*

\* The [[TweetStateRecord]] type is determined by the lkey of a tweet manhattan record:

\* metadata/delete\_state -> HardDeleted

\* metadata/soft\_delete\_state -> SoftDeleted

\* metadata/undelete\_state -> Undeleted

\* metadata/force\_added\_state -> ForceAdded

\*

\* See the README in this directory for more details about the state of a tweet.

\*/

sealed trait TweetStateRecord {

def tweetId: TweetId

def createdAt: Long

def stateKey: TweetKey.LKey.StateKey

def values: Map[String, Long] = Map("timestamp" -> createdAt)

def name: String

def toTweetMhRecord: TweetManhattanRecord = {

val valByteBuffer = ByteArrayCodec.toByteBuffer(Json.encode(values))

val value = ManhattanValue(valByteBuffer, Some(Time.fromMilliseconds(createdAt)))

TweetManhattanRecord(TweetKey(tweetId, stateKey), value)

}

}

object TweetStateRecord {

/\*\* When a soft-deleted or bounce deleted tweet is ultimately hard-deleted by an offline job. \*/

case class HardDeleted(tweetId: TweetId, createdAt: Long, deletedAt: Long)

extends TweetStateRecord {

// timestamp in the mh backend is the hard deletion timestamp

override def values = Map("timestamp" -> createdAt, "softdelete\_timestamp" -> deletedAt)

def stateKey = TweetKey.LKey.HardDeletionStateKey

def name = "hard\_deleted"

}

/\*\* When a tweet is deleted by the user. It can still be undeleted while in the soft deleted state. \*/

case class SoftDeleted(tweetId: TweetId, createdAt: Long) extends TweetStateRecord {

def stateKey = TweetKey.LKey.SoftDeletionStateKey

def name = "soft\_deleted"

}

/\*\* When a tweet is deleted by go/bouncer for violating Twitter Rules. It MAY NOT be undeleted. \*/

case class BounceDeleted(tweetId: TweetId, createdAt: Long) extends TweetStateRecord {

def stateKey = TweetKey.LKey.BounceDeletionStateKey

def name = "bounce\_deleted"

}

/\*\* When a tweet is undeleted by an internal system. \*/

case class Undeleted(tweetId: TweetId, createdAt: Long) extends TweetStateRecord {

def stateKey = TweetKey.LKey.UnDeletionStateKey

def name = "undeleted"

}

/\*\* When a tweet is created using the forceAdd endpoint. \*/

case class ForceAdded(tweetId: TweetId, createdAt: Long) extends TweetStateRecord {

def stateKey = TweetKey.LKey.ForceAddedStateKey

def name = "force\_added"

}

def fromTweetMhRecord(record: TweetManhattanRecord): Option[TweetStateRecord] = {

def ts = TimestampDecoder.decode(record, TimestampType.Default).getOrElse(0L)

def sdts = TimestampDecoder.decode(record, TimestampType.SoftDelete).getOrElse(0L)

def tweetId = record.pkey

record.lkey match {

case TweetKey.LKey.HardDeletionStateKey => Some(HardDeleted(tweetId, ts, sdts))

case TweetKey.LKey.SoftDeletionStateKey => Some(SoftDeleted(tweetId, ts))

case TweetKey.LKey.BounceDeletionStateKey => Some(BounceDeleted(tweetId, ts))

case TweetKey.LKey.UnDeletionStateKey => Some(Undeleted(tweetId, ts))

case TweetKey.LKey.ForceAddedStateKey => Some(ForceAdded(tweetId, ts))

case \_ => None

}

}

def fromTweetMhRecords(records: Seq[TweetManhattanRecord]): Seq[TweetStateRecord] =

records.flatMap(fromTweetMhRecord)

def mostRecent(records: Seq[TweetManhattanRecord]): Option[TweetStateRecord] =

fromTweetMhRecords(records).sortBy(\_.createdAt).lastOption

}