package com.twitter.recos.user\_tweet\_entity\_graph

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.graphjet.algorithms.counting.tweet.{

TweetMetadataRecommendationInfo,

TweetRecommendationInfo

}

import com.twitter.recos.recos\_common.thriftscala.{SocialProof, SocialProofType}

import scala.collection.JavaConverters.\_

class SocialProofHydrator(statsReceiver: StatsReceiver) {

private val stats = statsReceiver.scope(this.getClass.getSimpleName)

private val socialProofsDup = stats.counter("socialProofsDup")

private val socialProofsUni = stats.counter("socialProofsUni")

private val socialProofByTypeDup = stats.counter("socialProofByTypeDup")

private val socialProofByTypeUni = stats.counter("socialProofByTypeUni")

// If the social proof type is favorite, there are cases that one user favs, unfavs and then favs the same tweet again.

// In this case, UTEG only returns one valid social proof. Note that GraphJet library compares the number of unique users

// with the minSocialProofThreshold, so the threshold checking logic is correct.

// If the social proof type is reply or quote, there are valid cases that one user replies the same tweet multiple times.

// GraphJet does not handle this deduping because this is Twitter specific logic.

def getSocialProofs(

socialProofType: SocialProofType,

users: Seq[Long],

metadata: Seq[Long]

): Seq[SocialProof] = {

if (socialProofType == SocialProofType.Favorite && users.size > 1 && users.size != users.distinct.size) {

socialProofsDup.incr()

val unique = users

.zip(metadata)

.foldLeft[Seq[(Long, Long)]](Nil) { (list, next) =>

{

val test = list find { \_.\_1 == next.\_1 }

if (test.isEmpty) next +: list else list

}

}

.reverse

unique.map { case (user, data) => SocialProof(user, Some(data)) }

} else {

socialProofsUni.incr()

users.zip(metadata).map { case (user, data) => SocialProof(user, Some(data)) }

}

}

// Extract and dedup social proofs from GraphJet. Only Favorite based social proof needs to dedup.

// Return the social proofs (userId, metadata) pair in SocialProof thrift objects.

def addTweetSocialProofs(

tweet: TweetRecommendationInfo

): Option[Map[SocialProofType, Seq[SocialProof]]] = {

Some(

tweet.getSocialProof.asScala.map {

case (socialProofType, socialProof) =>

val socialProofThriftType = SocialProofType(socialProofType.toByte)

(

socialProofThriftType,

getSocialProofs(

socialProofThriftType,

socialProof.getConnectingUsers.asScala.map(\_.toLong),

socialProof.getMetadata.asScala.map(\_.toLong)

)

)

}.toMap

)

}

def getSocialProofs(users: Seq[Long]): Seq[Long] = {

if (users.size > 1) {

val distinctUsers = users.distinct

if (users.size != distinctUsers.size) {

socialProofByTypeDup.incr()

} else {

socialProofByTypeUni.incr()

}

distinctUsers

} else {

socialProofByTypeUni.incr()

users

}

}

// Extract and dedup social proofs from GraphJet. All social proof types need to dedup.

// Return the userId social proofs without metadata.

def addTweetSocialProofByType(tweet: TweetRecommendationInfo): Map[SocialProofType, Seq[Long]] = {

tweet.getSocialProof.asScala.map {

case (socialProofType, socialProof) =>

(

SocialProofType(socialProofType.toByte),

getSocialProofs(socialProof.getConnectingUsers.asScala.map(\_.toLong))

)

}.toMap

}

// The Hashtag and URL Social Proof. Dedup is not necessary.

def addMetadataSocialProofByType(

tweetMetadataRec: TweetMetadataRecommendationInfo

): Map[SocialProofType, Map[Long, Seq[Long]]] = {

tweetMetadataRec.getSocialProof.asScala.map {

case (socialProofType, socialProof) =>

(

SocialProofType(socialProofType.toByte),

socialProof.asScala.map {

case (authorId, tweetIds) =>

(authorId.toLong, tweetIds.asScala.map(\_.toLong))

}.toMap)

}.toMap

}

}