package com.twitter.usersignalservice.signals

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.simclusters\_v2.common.UserId

import com.twitter.simclusters\_v2.thriftscala.InternalId

import com.twitter.strato.client.Client

import com.twitter.strato.data.Conv

import com.twitter.strato.thrift.ScroogeConv

import com.twitter.twistly.thriftscala.RecentNegativeEngagedTweet

import com.twitter.twistly.thriftscala.TweetNegativeEngagementType

import com.twitter.twistly.thriftscala.UserRecentNegativeEngagedTweets

import com.twitter.usersignalservice.base.Query

import com.twitter.usersignalservice.base.StratoSignalFetcher

import com.twitter.usersignalservice.thriftscala.Signal

import com.twitter.usersignalservice.thriftscala.SignalType

import com.twitter.util.Future

import com.twitter.util.Timer

import javax.inject.Inject

import javax.inject.Singleton

@Singleton

case class NegativeEngagedTweetFetcher @Inject() (

stratoClient: Client,

timer: Timer,

stats: StatsReceiver)

extends StratoSignalFetcher[(UserId, Long), Unit, UserRecentNegativeEngagedTweets] {

import NegativeEngagedTweetFetcher.\_

override type RawSignalType = RecentNegativeEngagedTweet

override val name: String = this.getClass.getCanonicalName

override val statsReceiver: StatsReceiver = stats.scope(name)

override val stratoColumnPath: String = stratoPath

override val stratoView: Unit = None

override protected val keyConv: Conv[(UserId, Long)] = Conv.ofType

override protected val viewConv: Conv[Unit] = Conv.ofType

override protected val valueConv: Conv[UserRecentNegativeEngagedTweets] =

ScroogeConv.fromStruct[UserRecentNegativeEngagedTweets]

override protected def toStratoKey(userId: UserId): (UserId, Long) = (userId, defaultVersion)

override protected def toRawSignals(

stratoValue: UserRecentNegativeEngagedTweets

): Seq[RecentNegativeEngagedTweet] = {

stratoValue.recentNegativeEngagedTweets

}

override def process(

query: Query,

rawSignals: Future[Option[Seq[RecentNegativeEngagedTweet]]]

): Future[Option[Seq[Signal]]] = {

rawSignals.map {

\_.map { signals =>

signals

.filter(signal => negativeEngagedTweetTypeFilter(query.signalType, signal))

.map { signal =>

Signal(

query.signalType,

signal.engagedAt,

Some(InternalId.TweetId(signal.tweetId))

)

}

.groupBy(\_.targetInternalId) // groupBy if there's duplicated authorIds

.mapValues(\_.maxBy(\_.timestamp))

.values

.toSeq

.sortBy(-\_.timestamp)

.take(query.maxResults.getOrElse(Int.MaxValue))

}

}

}

}

object NegativeEngagedTweetFetcher {

val stratoPath = "recommendations/twistly/userRecentNegativeEngagedTweets"

private val defaultVersion = 0L

private def negativeEngagedTweetTypeFilter(

signalType: SignalType,

signal: RecentNegativeEngagedTweet

): Boolean = {

signalType match {

case SignalType.TweetDontLike =>

signal.engagementType == TweetNegativeEngagementType.DontLike

case SignalType.TweetSeeFewer =>

signal.engagementType == TweetNegativeEngagementType.SeeFewer

case SignalType.TweetReport =>

signal.engagementType == TweetNegativeEngagementType.ReportClick

case SignalType.NegativeEngagedTweetId => true

case \_ => false

}

}

}