package com.twitter.recosinjector.edges

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.frigate.common.store.TweetCreationTimeMHStore

import com.twitter.frigate.common.util.SnowflakeUtils

import com.twitter.recos.internal.thriftscala.RecosUserTweetInfo

import com.twitter.recos.internal.thriftscala.TweetType

import com.twitter.recos.util.Action

import com.twitter.recosinjector.decider.RecosInjectorDecider

import com.twitter.recosinjector.decider.RecosInjectorDeciderConstants

import com.twitter.recosinjector.util.TweetCreateEventDetails

import com.twitter.util.Future

import com.twitter.util.Time

class TweetEventToUserTweetGraphBuilder(

userTweetEntityEdgeBuilder: UserTweetEntityEdgeBuilder,

tweetCreationStore: TweetCreationTimeMHStore,

decider: RecosInjectorDecider

)(

override implicit val statsReceiver: StatsReceiver)

extends EventToMessageBuilder[TweetCreateEventDetails, UserTweetEntityEdge] {

private val numRetweetEdgesCounter = statsReceiver.counter("num\_retweet\_edge")

private val numIsDecider = statsReceiver.counter("num\_decider\_enabled")

private val numIsNotDecider = statsReceiver.counter("num\_decider\_not\_enabled")

override def shouldProcessEvent(event: TweetCreateEventDetails): Future[Boolean] = {

val isDecider = decider.isAvailable(

RecosInjectorDeciderConstants.TweetEventTransformerUserTweetEntityEdgesDecider

)

if (isDecider) {

numIsDecider.incr()

Future(true)

} else {

numIsNotDecider.incr()

Future(false)

}

}

/\*\*

\* Build a Retweet edge: author -> RT -> SourceTweetId.

\*/

private def buildRetweetEdge(event: TweetCreateEventDetails) = {

val userTweetEngagement = event.userTweetEngagement

val tweetId = userTweetEngagement.tweetId

event.sourceTweetDetails

.map { sourceTweetDetails =>

val sourceTweetId = sourceTweetDetails.tweet.id // Id of the tweet being Retweeted

val sourceTweetEntitiesMapFut = userTweetEntityEdgeBuilder.getEntitiesMapAndUpdateCache(

tweetId = sourceTweetId,

tweetDetails = Some(sourceTweetDetails)

)

sourceTweetEntitiesMapFut.map { sourceTweetEntitiesMap =>

val edge = UserTweetEntityEdge(

sourceUser = userTweetEngagement.engageUserId,

targetTweet = sourceTweetId,

action = Action.Retweet,

metadata = Some(tweetId), // metadata is the tweetId

cardInfo = Some(sourceTweetDetails.cardInfo.toByte),

entitiesMap = sourceTweetEntitiesMap,

tweetDetails = Some(sourceTweetDetails)

)

numRetweetEdgesCounter.incr()

Seq(edge)

}

}.getOrElse(Future.Nil)

}

override def buildEdges(event: TweetCreateEventDetails): Future[Seq[UserTweetEntityEdge]] = {

val userTweetEngagement = event.userTweetEngagement

userTweetEngagement.action match {

case Action.Retweet =>

buildRetweetEdge(event)

case \_ =>

Future.Nil

}

}

override def filterEdges(

event: TweetCreateEventDetails,

edges: Seq[UserTweetEntityEdge]

): Future[Seq[UserTweetEntityEdge]] = {

Future(edges) // No filtering for now. Add more if needed

}

}