package com.twitter.tweetypie.repository

import com.twitter.audience\_rewards.thriftscala.HasSuperFollowingRelationshipRequest

import com.twitter.stitch.Stitch

import com.twitter.strato.client.Fetcher

import com.twitter.strato.client.{Client => StratoClient}

import com.twitter.tweetypie.Future

import com.twitter.tweetypie.UserId

import com.twitter.tweetypie.core.TweetCreateFailure

import com.twitter.tweetypie.thriftscala.ExclusiveTweetControl

import com.twitter.tweetypie.thriftscala.TweetCreateState

object StratoSuperFollowRelationsRepository {

type Type = (UserId, UserId) => Stitch[Boolean]

def apply(client: StratoClient): Type = {

val column = "audiencerewards/superFollows/hasSuperFollowingRelationshipV2"

val fetcher: Fetcher[HasSuperFollowingRelationshipRequest, Unit, Boolean] =

client.fetcher[HasSuperFollowingRelationshipRequest, Boolean](column)

(authorId, viewerId) => {

// Owner of an exclusive tweet chain can respond to their own

// tweets / replies, despite not super following themselves

if (authorId == viewerId) {

Stitch.True

} else {

val key = HasSuperFollowingRelationshipRequest(authorId, viewerId)

// The default relation for this column is "missing", aka None.

// This needs to be mapped to false since Super Follows are a sparse relation.

fetcher.fetch(key).map(\_.v.getOrElse(false))

}

}

}

object Validate {

def apply(

exclusiveTweetControl: Option[ExclusiveTweetControl],

userId: UserId,

superFollowRelationsRepo: StratoSuperFollowRelationsRepository.Type

): Future[Unit] = {

Stitch

.run {

exclusiveTweetControl.map(\_.conversationAuthorId) match {

// Don't do exclusive tweet validation on non exclusive tweets.

case None =>

Stitch.value(true)

case Some(convoAuthorId) =>

superFollowRelationsRepo(userId, convoAuthorId)

}

}.map {

case true => Future.Unit

case false =>

Future.exception(TweetCreateFailure.State(TweetCreateState.SourceTweetNotFound))

}.flatten

}

}

}