package com.twitter.tweetypie.federated.columns

import com.twitter.spam.rtf.thriftscala.SafetyLevel

import com.twitter.stitch.MapGroup

import com.twitter.tweetypie.UserId

import com.twitter.tweetypie.federated.columns.FederatedFieldGroupBuilder.allCountFields

import com.twitter.tweetypie.federated.columns.FederatedFieldGroupBuilder.countTweetFields

import com.twitter.tweetypie.thriftscala.GetTweetFieldsOptions

import com.twitter.tweetypie.thriftscala.GetTweetFieldsRequest

import com.twitter.tweetypie.thriftscala.GetTweetFieldsResult

import com.twitter.tweetypie.thriftscala.StatusCounts

import com.twitter.tweetypie.thriftscala.Tweet

import com.twitter.tweetypie.thriftscala.TweetInclude

import com.twitter.util.Future

import com.twitter.util.Throw

import com.twitter.util.Try

case class GroupOptions(twitterUserId: Option[UserId])

object FederatedFieldGroupBuilder {

type Type = GroupOptions => MapGroup[FederatedFieldReq, GetTweetFieldsResult]

def apply(

getTweetFieldsHandler: GetTweetFieldsRequest => Future[Seq[GetTweetFieldsResult]]

): Type = {

FederatedFieldGroup(getTweetFieldsHandler, \_)

}

// The set of non-deprecated count field includes

val allCountFields: Set[TweetInclude] = Set(

TweetInclude.CountsFieldId(StatusCounts.RetweetCountField.id),

TweetInclude.CountsFieldId(StatusCounts.QuoteCountField.id),

TweetInclude.CountsFieldId(StatusCounts.FavoriteCountField.id),

TweetInclude.CountsFieldId(StatusCounts.ReplyCountField.id),

TweetInclude.CountsFieldId(StatusCounts.BookmarkCountField.id),

)

// Tweet field includes which contain counts. These are the only fields where count field includes are relevant.

val countTweetFields: Set[TweetInclude] = Set(

TweetInclude.TweetFieldId(Tweet.CountsField.id),

TweetInclude.TweetFieldId(Tweet.PreviousCountsField.id))

}

case class FederatedFieldGroup(

getTweetFieldsHandler: GetTweetFieldsRequest => Future[Seq[GetTweetFieldsResult]],

options: GroupOptions)

extends MapGroup[FederatedFieldReq, GetTweetFieldsResult] {

override protected def run(

reqs: Seq[FederatedFieldReq]

): Future[FederatedFieldReq => Try[GetTweetFieldsResult]] = {

// requesting the field ids of the requested additional field ids in this group

val fieldIncludes: Set[TweetInclude] = reqs.map { req: FederatedFieldReq =>

TweetInclude.TweetFieldId(req.fieldId)

}.toSet

val allIncludes: Set[TweetInclude] = if (fieldIncludes.intersect(countTweetFields).nonEmpty) {

// if counts are being requested we include all count fields by default

// because there is no way to specify them individually with federated fields,

fieldIncludes ++ allCountFields

} else {

fieldIncludes

}

val gtfOptions = GetTweetFieldsOptions(

tweetIncludes = allIncludes,

forUserId = options.twitterUserId,

// visibility filtering happens at the api layer / tweet top level

// and therefore is not required at individual field level

safetyLevel = Some(SafetyLevel.FilterNone)

)

getTweetFieldsHandler(

GetTweetFieldsRequest(

tweetIds = reqs.map(\_.tweetId).distinct,

options = gtfOptions

)

).map {

response =>

{ req =>

response.find(\_.tweetId == req.tweetId) match {

case Some(result) => Try(result)

case None =>

Throw(new NoSuchElementException(s"response not found for tweet: ${req.tweetId}"))

}

}

}

}

}