package com.twitter.tweetypie.federated.columns

import com.twitter.stitch.MapGroup

import com.twitter.stitch.Stitch

import com.twitter.strato.access.Access.LdapGroup

import com.twitter.strato.catalog.Fetch

import com.twitter.strato.catalog.OpMetadata

import com.twitter.strato.config.AnyOf

import com.twitter.strato.config.ContactInfo

import com.twitter.strato.config.FromColumns

import com.twitter.strato.config.Has

import com.twitter.strato.config.Path

import com.twitter.strato.config.Policy

import com.twitter.strato.data.Conv

import com.twitter.strato.data.Description.PlainText

import com.twitter.strato.data.Lifecycle.Production

import com.twitter.strato.fed.StratoFed

import com.twitter.strato.response.Err

import com.twitter.strato.thrift.ScroogeConv

import com.twitter.tweetypie.{thriftscala => thrift}

import com.twitter.tweetypie.TweetId

import com.twitter.tweetypie.thriftscala.federated.GetStoredTweetsView

import com.twitter.tweetypie.thriftscala.federated.GetStoredTweetsResponse

import com.twitter.util.Future

import com.twitter.util.Return

import com.twitter.util.Throw

import com.twitter.util.Try

class GetStoredTweetsColumn(

getStoredTweets: thrift.GetStoredTweetsRequest => Future[Seq[thrift.GetStoredTweetsResult]])

extends StratoFed.Column(GetStoredTweetsColumn.Path)

with StratoFed.Fetch.Stitch {

override val contactInfo: ContactInfo = TweetypieContactInfo

override val metadata: OpMetadata = OpMetadata(

lifecycle = Some(Production),

description = Some(PlainText("Fetches hydrated Tweets regardless of Tweet state."))

)

override val policy: Policy = AnyOf(

Seq(

FromColumns(

Set(

Path("tweetypie/data-provider/storedTweets.User"),

Path("note\_tweet/data-provider/noteTweetForZipbird.User"))),

Has(LdapGroup("tweetypie-team"))

))

override type Key = TweetId

override type View = GetStoredTweetsView

override type Value = GetStoredTweetsResponse

override val keyConv: Conv[Key] = Conv.ofType

override val viewConv: Conv[View] = ScroogeConv.fromStruct[GetStoredTweetsView]

override val valueConv: Conv[Value] = ScroogeConv.fromStruct[GetStoredTweetsResponse]

override def fetch(key: Key, view: View): Stitch[Result[Value]] = {

Stitch.call(key, Group(view))

}

private case class Group(view: GetStoredTweetsView)

extends MapGroup[TweetId, Fetch.Result[GetStoredTweetsResponse]] {

override protected def run(

keys: Seq[TweetId]

): Future[TweetId => Try[Result[GetStoredTweetsResponse]]] = {

val options = thrift.GetStoredTweetsOptions(

bypassVisibilityFiltering = view.bypassVisibilityFiltering,

forUserId = view.forUserId,

additionalFieldIds = view.additionalFieldIds

)

getStoredTweets(thrift.GetStoredTweetsRequest(keys, Some(options)))

.map(transformAndGroupByTweetId)

.handle {

case \_ =>

\_ => Throw[Result[GetStoredTweetsResponse]](Err(Err.Internal))

}

}

private def transformAndGroupByTweetId(

results: Seq[thrift.GetStoredTweetsResult]

): Map[TweetId, Try[Fetch.Result[GetStoredTweetsResponse]]] = {

results

.map(result => GetStoredTweetsResponse(result.storedTweet))

.groupBy(\_.storedTweet.tweetId)

.map {

case (tweetId, Seq(result)) => (tweetId, Return(Fetch.Result.found(result)))

case (tweetId, multipleResults) =>

(

tweetId,

Throw(Err(Err.BadRequest, s"Got ${multipleResults.size} results for $tweetId")))

}

}

}

}

object GetStoredTweetsColumn {

val Path = "tweetypie/internal/getStoredTweets.Tweet"

}