package com.twitter.tweetypie.federated.columns

import com.twitter.stitch.Stitch

import com.twitter.strato.catalog.OpMetadata

import com.twitter.strato.config.ContactInfo

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.opcontext.OpContext

import com.twitter.strato.thrift.ScroogeConv

import com.twitter.tweetypie.federated.context.GetRequestContext

import com.twitter.tweetypie.federated.prefetcheddata.PrefetchedDataResponse

import com.twitter.tweetypie.thriftscala.TweetDeleteState

import com.twitter.tweetypie.thriftscala.{graphql => gql}

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

import com.twitter.util.Future

class DeleteTweetColumn(

deleteTweet: thrift.DeleteTweetsRequest => Future[Seq[thrift.DeleteTweetResult]],

getRequestContext: GetRequestContext,

) extends StratoFed.Column(DeleteTweetColumn.Path)

with StratoFed.Execute.StitchWithContext

with StratoFed.HandleDarkRequests {

override val policy: Policy = AccessPolicy.TweetMutationCommonAccessPolicies

override val isIdempotent: Boolean = true

override type Arg = gql.DeleteTweetRequest

override type Result = gql.DeleteTweetResponseWithSubqueryPrefetchItems

override val argConv: Conv[Arg] = ScroogeConv.fromStruct

override val resultConv: Conv[Result] = ScroogeConv.fromStruct

override val contactInfo: ContactInfo = TweetypieContactInfo

override val metadata: OpMetadata =

OpMetadata(Some(Production), Some(PlainText("Deletes a tweet by the calling Twitter user.")))

override def execute(request: Arg, opContext: OpContext): Stitch[Result] = {

val ctx = getRequestContext(opContext)

val thriftDeleteTweetRequest = thrift.DeleteTweetsRequest(

tweetIds = Seq(request.tweetId),

// byUserId is picked up by the context in tweetypie.deleteTweet,

// but we're passing it in here to be explicit

byUserId = Some(ctx.twitterUserId),

)

val stitchDeleteTweet = handleDarkRequest(opContext)(

light = {

Stitch.callFuture(deleteTweet(thriftDeleteTweetRequest))

},

// For dark requests, we don't want to send traffic to tweetypie.

// Since the response is the same regardless of the request, we take a no-op

// action instead.

dark = Stitch.value(Seq(thrift.DeleteTweetResult(request.tweetId, TweetDeleteState.Ok)))

)

stitchDeleteTweet.map { result: Seq[thrift.DeleteTweetResult] =>

result.headOption match {

case Some(thrift.DeleteTweetResult(id, TweetDeleteState.Ok)) =>

gql.DeleteTweetResponseWithSubqueryPrefetchItems(

data = Some(gql.DeleteTweetResponse(Some(id))),

// Prefetch data is always NotFound to prevent subqueries from hydrating via weaverbird

// and possibly returning inconsistent results, i.e. a Found tweet.

subqueryPrefetchItems = Some(PrefetchedDataResponse.notFound(id).value)

)

case Some(thrift.DeleteTweetResult(\_, TweetDeleteState.PermissionError)) =>

throw ApiErrors.DeletePermissionErr

case \_ =>

throw ApiErrors.GenericAccessDeniedErr

}

}

}

}

object DeleteTweetColumn {

val Path = "tweetypie/deleteTweet.Tweet"

}