package com.twitter.tweetypie

package federated.columns

import com.twitter.io.Buf

import com.twitter.scrooge.TFieldBlob

import com.twitter.stitch.Stitch

import com.twitter.strato.access.Access

import com.twitter.strato.catalog.OpMetadata

import com.twitter.strato.config.AllowAll

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.data.Type

import com.twitter.strato.data.Val

import com.twitter.strato.fed.StratoFed

import com.twitter.strato.opcontext.OpContext

import com.twitter.strato.serialization.MVal

import com.twitter.strato.serialization.Thrift

import com.twitter.strato.util.Strings

import com.twitter.tweetypie.thriftscala.GetTweetFieldsResult

import com.twitter.tweetypie.thriftscala.SetAdditionalFieldsRequest

import com.twitter.tweetypie.thriftscala.Tweet

import com.twitter.tweetypie.thriftscala.TweetFieldsResultState.Found

import com.twitter.util.Future

import org.apache.thrift.protocol.TField

/\*\*

\* Federated strato column to return tweet fields

\* @param federatedFieldsGroup Group to be used for Stitch batching.

\* This is a function that takes a GroupOptions and returns a FederatedFieldGroup.

\* Using a function that accepts a GroupOptions allows for Stitch to handle a new group for distinct GroupOptions.

\* @param setAdditionalFields Handler to set additional fields on tweets.

\* @param stratoValueType Type to be returned by the strato column.

\* @param tfield Tweet thrift field to be stored

\* @param pathName Path to be used in the strato catalog

\*/

class FederatedFieldColumn(

federatedFieldsGroup: FederatedFieldGroupBuilder.Type,

setAdditionalFields: SetAdditionalFieldsRequest => Future[Unit],

stratoValueType: Type,

tfield: TField,

pathOverride: Option[String] = None)

extends StratoFed.Column(pathOverride.getOrElse(FederatedFieldColumn.makeColumnPath(tfield)))

with StratoFed.Fetch.StitchWithContext

with StratoFed.Put.Stitch {

type Key = Long

type View = Unit

type Value = Val.T

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

override val viewConv: Conv[View] = Conv.ofType

override val valueConv: Conv[Value] = Conv(stratoValueType, identity, identity)

override val policy: Policy = AllowAll

/\*

\* A fetch that proxies GetTweetFieldsColumn.fetch but only requests and

\* returns one specific field.

\*/

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

val twitterUserId: Option[UserId] = Access.getTwitterUserId match {

// Access.getTwitterUserId should return a value when request is made on behalf of a user

// and will not return a value otherwise

case Some(twitterUser) => Some(twitterUser.id)

case None => None

}

val stitchGroup = federatedFieldsGroup(GroupOptions(twitterUserId))

Stitch

.call(FederatedFieldReq(tweetId, tfield.id), stitchGroup).map {

result: GetTweetFieldsResult =>

result.tweetResult match {

case Found(f) =>

f.tweet.getFieldBlob(tfield.id) match {

case Some(v: TFieldBlob) =>

found(blobToVal(v))

case None => missing

}

case \_ => missing

}

}

}

/\*

\* A strato put interface for writing a single additional field to a tweet

\*/

override def put(tweetId: Key, value: Val.T): Stitch[Unit] = {

val tweet: Tweet = Tweet(id = tweetId).setField(valToBlob(value))

val request: SetAdditionalFieldsRequest = SetAdditionalFieldsRequest(tweet)

Stitch.callFuture(setAdditionalFields(request))

}

val mval: Thrift.Codec = MVal.codec(stratoValueType).thrift(4)

def valToBlob(value: Val.T): TFieldBlob =

TFieldBlob(tfield, mval.write[Buf](value, Thrift.compactProto))

def blobToVal(thriftFieldBlob: TFieldBlob): Val.T =

mval.read(thriftFieldBlob.content, Thrift.compactProto)

override val contactInfo: ContactInfo = TweetypieContactInfo

override val metadata: OpMetadata = OpMetadata(

lifecycle = Some(Production),

description = Some(PlainText(s"A federated column for the field tweet.$stratoValueType"))

)

}

object FederatedFieldColumn {

val idAllowlist: Seq[Short] = Seq(

Tweet.CoreDataField.id,

Tweet.LanguageField.id,

Tweet.ConversationMutedField.id

)

val ID\_START = 157

val ID\_END = 32000

private val MigrationFields: Seq[Short] = Seq(157)

def isFederatedField(id: Short) = id >= ID\_START && id < ID\_END || idAllowlist.contains(id)

def isMigrationFederatedField(tField: TField): Boolean = MigrationFields.contains(tField.id)

/\* federated field column strato configs must conform to this

\* path name scheme for tweetypie to pick them up

\*/

def makeColumnPath(tField: TField) = {

val columnName = Strings.toCamelCase(tField.name.stripSuffix("id"))

s"tweetypie/fields/${columnName}.Tweet"

}

def makeV1ColumnPath(tField: TField): String = {

val columnName = Strings.toCamelCase(tField.name.stripSuffix("id"))

s"tweetypie/fields/$columnName-V1.Tweet"

}

}