package com.twitter.tweetypie.storage

import com.twitter.bijection.Injection

import com.twitter.io.Buf

import com.twitter.stitch.Stitch

import com.twitter.storage.client.manhattan.bijections.Bijections.BufInjection

import com.twitter.storage.client.manhattan.kv.ManhattanKVEndpoint

import com.twitter.storage.client.manhattan.kv.impl.DescriptorP1L1

import com.twitter.storage.client.manhattan.kv.impl.Component

import com.twitter.storage.client.manhattan.kv.{impl => mh}

import com.twitter.storage.client.manhattan.bijections.Bijections.StringInjection

import com.twitter.util.Time

import java.nio.ByteBuffer

import scala.util.control.NonFatal

case class TweetManhattanRecord(key: TweetKey, value: TweetManhattanValue) {

def pkey: TweetId = key.tweetId

def lkey: TweetKey.LKey = key.lKey

/\*\*

\* Produces a representation that is human-readable, but contains

\* all of the information from the record. It is not intended for

\* producing machine-readable values.

\*

\* This conversion is relatively expensive, so beware of using it in

\* hot code paths.

\*/

override def toString: String = {

val valueString =

try {

key.lKey match {

case \_: TweetKey.LKey.MetadataKey =>

StringCodec.fromByteBuffer(value.contents)

case \_: TweetKey.LKey.FieldKey =>

val tFieldBlob = TFieldBlobCodec.fromByteBuffer(value.contents)

s"TFieldBlob(${tFieldBlob.field}, 0x${Buf.slowHexString(tFieldBlob.content)})"

case TweetKey.LKey.Unknown(\_) =>

"0x" + Buf.slowHexString(Buf.ByteBuffer.Shared(value.contents))

}

} catch {

case NonFatal(e) =>

val hexValue = Buf.slowHexString(Buf.ByteBuffer.Shared(value.contents))

s"0x$hexValue (failed to decode due to $e)"

}

s"$key => ${value.copy(contents = valueString)}"

}

}

object ManhattanOperations {

type Read = TweetId => Stitch[Seq[TweetManhattanRecord]]

type Insert = TweetManhattanRecord => Stitch[Unit]

type Delete = (TweetKey, Option[Time]) => Stitch[Unit]

type DeleteRange = TweetId => Stitch[Unit]

object PkeyInjection extends Injection[TweetId, String] {

override def apply(tweetId: TweetId): String = TweetKey.padTweetIdStr(tweetId)

override def invert(str: String): scala.util.Try[TweetId] = scala.util.Try(str.toLong)

}

case class InvalidLkey(lkeyStr: String) extends Exception

object LkeyInjection extends Injection[TweetKey.LKey, String] {

override def apply(lkey: TweetKey.LKey): String = lkey.toString

override def invert(str: String): scala.util.Try[TweetKey.LKey] =

scala.util.Success(TweetKey.LKey.fromString(str))

}

val KeyDescriptor: DescriptorP1L1.EmptyKey[TweetId, TweetKey.LKey] =

mh.KeyDescriptor(

Component(PkeyInjection.andThen(StringInjection)),

Component(LkeyInjection.andThen(StringInjection))

)

val ValueDescriptor: mh.ValueDescriptor.EmptyValue[ByteBuffer] = mh.ValueDescriptor(BufInjection)

}

class ManhattanOperations(dataset: String, mhEndpoint: ManhattanKVEndpoint) {

import ManhattanOperations.\_

private[this] def pkey(tweetId: TweetId) = KeyDescriptor.withDataset(dataset).withPkey(tweetId)

def read: Read = { tweetId =>

mhEndpoint.slice(pkey(tweetId).under(), ValueDescriptor).map { mhData =>

mhData.map {

case (key, value) => TweetManhattanRecord(TweetKey(key.pkey, key.lkey), value)

}

}

}

def insert: Insert =

record => {

val mhKey = pkey(record.key.tweetId).withLkey(record.key.lKey)

mhEndpoint.insert(mhKey, ValueDescriptor.withValue(record.value))

}

def delete: Delete = (key, time) => mhEndpoint.delete(pkey(key.tweetId).withLkey(key.lKey), time)

def deleteRange: DeleteRange =

tweetId => mhEndpoint.deleteRange(KeyDescriptor.withDataset(dataset).withPkey(tweetId).under())

}