package com.twitter.simclusters\_v2.stores

import com.twitter.simclusters\_v2.common.SimClustersEmbedding

import com.twitter.simclusters\_v2.common.SimClustersMultiEmbeddingId.\_

import com.twitter.simclusters\_v2.thriftscala.{

SimClustersMultiEmbedding,

SimClustersEmbeddingId,

SimClustersMultiEmbeddingId

}

import com.twitter.storehaus.ReadableStore

import com.twitter.util.Future

/\*\*

\* The helper methods for SimClusters Multi-Embedding based ReadableStore

\*/

object SimClustersMultiEmbeddingStore {

/\*\*

\* Only support the Values based Multi-embedding transformation.

\*/

case class SimClustersMultiEmbeddingWrapperStore(

sourceStore: ReadableStore[SimClustersMultiEmbeddingId, SimClustersMultiEmbedding])

extends ReadableStore[SimClustersEmbeddingId, SimClustersEmbedding] {

override def get(k: SimClustersEmbeddingId): Future[Option[SimClustersEmbedding]] = {

sourceStore.get(toMultiEmbeddingId(k)).map(\_.map(toSimClustersEmbedding(k, \_)))

}

// Override the multiGet for better batch performance.

override def multiGet[K1 <: SimClustersEmbeddingId](

ks: Set[K1]

): Map[K1, Future[Option[SimClustersEmbedding]]] = {

if (ks.isEmpty) {

Map.empty

} else {

// Aggregate multiple get requests by MultiEmbeddingId

val multiEmbeddingIds = ks.map { k =>

k -> toMultiEmbeddingId(k)

}.toMap

val multiEmbeddings = sourceStore.multiGet(multiEmbeddingIds.values.toSet)

ks.map { k =>

k -> multiEmbeddings(multiEmbeddingIds(k)).map(\_.map(toSimClustersEmbedding(k, \_)))

}.toMap

}

}

private def toSimClustersEmbedding(

id: SimClustersEmbeddingId,

multiEmbedding: SimClustersMultiEmbedding

): SimClustersEmbedding = {

multiEmbedding match {

case SimClustersMultiEmbedding.Values(values) =>

val subId = toSubId(id)

if (subId >= values.embeddings.size) {

throw new IllegalArgumentException(

s"SimClustersMultiEmbeddingId $id is over the size of ${values.embeddings.size}")

} else {

values.embeddings(subId).embedding

}

case \_ =>

throw new IllegalArgumentException(

s"Invalid SimClustersMultiEmbedding $id, $multiEmbedding")

}

}

}

def toSimClustersEmbeddingStore(

sourceStore: ReadableStore[SimClustersMultiEmbeddingId, SimClustersMultiEmbedding]

): ReadableStore[SimClustersEmbeddingId, SimClustersEmbedding] = {

SimClustersMultiEmbeddingWrapperStore(sourceStore)

}

}