package com.twitter.ann.common

import com.twitter.stitch.Stitch

/\*\*

\* Implementation of QueryableById that composes an EmbeddingProducer and a Queryable so that we

\* can get nearest neighbors given an id of type T1

\* @param embeddingProducer provides an embedding given an id.

\* @param queryable provides a list of neighbors given an embedding.

\* @tparam T1 type of the query.

\* @tparam T2 type of the result.

\* @tparam P runtime parameters supported by the index.

\* @tparam D distance function used in the index.

\*/

class QueryableByIdImplementation[T1, T2, P <: RuntimeParams, D <: Distance[D]](

embeddingProducer: EmbeddingProducer[T1],

queryable: Queryable[T2, P, D])

extends QueryableById[T1, T2, P, D] {

override def queryById(

id: T1,

numOfNeighbors: Int,

runtimeParams: P

): Stitch[List[T2]] = {

embeddingProducer.produceEmbedding(id).flatMap { embeddingOption =>

embeddingOption

.map { embedding =>

Stitch.callFuture(queryable.query(embedding, numOfNeighbors, runtimeParams))

}.getOrElse {

Stitch.value(List.empty)

}

}

}

override def queryByIdWithDistance(

id: T1,

numOfNeighbors: Int,

runtimeParams: P

): Stitch[List[NeighborWithDistance[T2, D]]] = {

embeddingProducer.produceEmbedding(id).flatMap { embeddingOption =>

embeddingOption

.map { embedding =>

Stitch.callFuture(queryable.queryWithDistance(embedding, numOfNeighbors, runtimeParams))

}.getOrElse {

Stitch.value(List.empty)

}

}

}

override def batchQueryById(

ids: Seq[T1],

numOfNeighbors: Int,

runtimeParams: P

): Stitch[List[NeighborWithSeed[T1, T2]]] = {

Stitch

.traverse(ids) { id =>

embeddingProducer.produceEmbedding(id).flatMap { embeddingOption =>

embeddingOption

.map { embedding =>

Stitch

.callFuture(queryable.query(embedding, numOfNeighbors, runtimeParams)).map(

\_.map(neighbor => NeighborWithSeed(id, neighbor)))

}.getOrElse {

Stitch.value(List.empty)

}.handle { case \_ => List.empty }

}

}.map { \_.toList.flatten }

}

override def batchQueryWithDistanceById(

ids: Seq[T1],

numOfNeighbors: Int,

runtimeParams: P

): Stitch[List[NeighborWithDistanceWithSeed[T1, T2, D]]] = {

Stitch

.traverse(ids) { id =>

embeddingProducer.produceEmbedding(id).flatMap { embeddingOption =>

embeddingOption

.map { embedding =>

Stitch

.callFuture(queryable.queryWithDistance(embedding, numOfNeighbors, runtimeParams))

.map(\_.map(neighbor =>

NeighborWithDistanceWithSeed(id, neighbor.neighbor, neighbor.distance)))

}.getOrElse {

Stitch.value(List.empty)

}.handle { case \_ => List.empty }

}

}.map {

\_.toList.flatten

}

}

}