package com.twitter.simclusters\_v2.scio.bq\_generation

package simclusters\_index\_generation

import com.spotify.scio.ScioContext

import com.spotify.scio.values.SCollection

import com.twitter.simclusters\_v2.scio.bq\_generation.common.BQGenerationUtil.getNSFWTweetIdDenylistSQL

import com.twitter.simclusters\_v2.scio.bq\_generation.common.BQGenerationUtil.getTweetIdWithFavCountSQL

import com.twitter.simclusters\_v2.scio.bq\_generation.common.BQGenerationUtil.getTweetIdWithMediaAndNSFWAuthorFilterSQL

import com.twitter.simclusters\_v2.scio.bq\_generation.common.BQGenerationUtil.getUserTweetEngagementEventPairSQL

import com.twitter.simclusters\_v2.scio.bq\_generation.common.BQGenerationUtil.generateClusterTopTweetIntersectionWithFavBasedIndexSQL

import com.twitter.simclusters\_v2.scio.bq\_generation.simclusters\_index\_generation.Config.simclustersEngagementBasedIndexGenerationSQLPath

import com.twitter.simclusters\_v2.scio.bq\_generation.common.IndexGenerationUtil.TopKTweetsForClusterKey

import com.twitter.simclusters\_v2.scio.bq\_generation.common.IndexGenerationUtil.parseClusterTopKTweetsFn

import com.twitter.wtf.beam.bq\_embedding\_export.BQQueryUtils

import org.apache.beam.sdk.io.gcp.bigquery.BigQueryIO

import org.joda.time.DateTime

object EngagementEventBasedClusterToTweetIndexFromBQ {

/\*

\* Reads the user-tweet-interaction table and apply tweet fav count filter

\* Returns the post processed table results in SQL string format

\*

\* Input:

\* - startTime: DateTime

\* The earliest timestamp from the user-tweet-interaction table

\* - endTime: DateTime

\* The latest timestamp from the user-tweet-interaction table

\* - minFavCount: Int

\* Whether we want to enable tweet fav count filters

\*

\* Return:

\* String - Post processed table results in SQL string format

\*/

def getTweetInteractionTableWithFavCountFilter(

startTime: DateTime,

endTime: DateTime,

minFavCount: Int

): String = {

if (minFavCount > 0) {

val tweetFavCountSQL = getTweetIdWithFavCountSQL(startTime, endTime)

s"""

| WITH tweet\_fav\_count AS (${tweetFavCountSQL})

| SELECT userId, tweetId, tsMillis

| FROM user\_tweet\_interaction\_with\_min\_interaction\_count\_filter

| JOIN tweet\_fav\_count

| USING(tweetId)

| WHERE tweet\_fav\_count.favCount >= ${minFavCount}

|""".stripMargin

} else {

// Directly read from the table without applying any filters

s"SELECT userId, tweetId, tsMillis FROM user\_tweet\_interaction\_with\_min\_interaction\_count\_filter"

}

}

/\*

\* Reads the user-tweet-interaction table and apply health and video filters if specified.

\* Returns the post processed table results in SQL string format

\*

\* Input:

\* - tableName: String

\* Schema of the table

\* userId: Long

\* tweetId: Long

\* tsMillis: Long

\* - startTime: DateTime

\* The earliest timestamp from the user-tweet-interaction table

\* - endTime: DateTime

\* The latest timestamp from the user-tweet-interaction table

\* - enableHealthAndVideoFilters: Boolean

\* Whether we want to enable health filters and video only filters

\*

\* Return:

\* String - Post processed table results in SQL string format

\*/

def getTweetInteractionTableWithHealthFilter(

startTime: DateTime,

endTime: DateTime,

enableHealthAndVideoFilters: Boolean,

): String = {

if (enableHealthAndVideoFilters) {

// Get SQL for tweets with media and NSFW filter

val tweetWithMediaAndNSFWAuthorFilterSQL = getTweetIdWithMediaAndNSFWAuthorFilterSQL(

startTime,

endTime,

filterMediaType = Some(3), // VideoTweets MediaType = 3

filterNSFWAuthor = true

)

// Get SQL for NSFW tweet id deny list

val nsfwTweetDenylistSQL = getNSFWTweetIdDenylistSQL(startTime, endTime)

// Combine the health filter SQLs

s"""

|SELECT userId, tweetId, tsMillis FROM user\_tweet\_interaction\_with\_fav\_count\_filter JOIN (

| ${tweetWithMediaAndNSFWAuthorFilterSQL}

| AND tweetId NOT IN (${nsfwTweetDenylistSQL})

|) USING(tweetId)

|""".stripMargin

} else {

// Directly read from the table without applying any filters

s"SELECT userId, tweetId, tsMillis FROM user\_tweet\_interaction\_with\_fav\_count\_filter"

}

}

def getTopKTweetsForClusterKeyBQ(

sc: ScioContext,

queryTimestamp: DateTime,

maxTweetAgeHours: Int,

consumerEmbeddingsSQL: String,

userTweetEngagementEventPairSqlPath: String,

userTweetEngagementEventPairTemplateVariable: Map[String, String],

enableHealthAndVideoFilters: Boolean,

enableFavClusterTopKTweetsIntersection: Boolean,

minInteractionCount: Int,

minFavCount: Int,

tweetEmbeddingsLength: Int,

tweetEmbeddingsHalfLife: Int,

minEngagementPerCluster: Int,

clusterTopKTweets: Int

): SCollection[TopKTweetsForClusterKey] = {

// Define template variables which we would like to be replaced in the corresponding sql file

val startTime = queryTimestamp.minusHours(maxTweetAgeHours)

val endTime = queryTimestamp

val indexGenerationTemplateVariables =

Map(

"HALF\_LIFE" -> tweetEmbeddingsHalfLife.toString,

"CURRENT\_TS" -> queryTimestamp.toString(),

"START\_TIME" -> startTime.toString(),

"END\_TIME" -> endTime.toString(),

"USER\_TWEET\_ENGAGEMENT\_TABLE\_SQL" ->

getUserTweetEngagementEventPairSQL(

startTime,

endTime,

userTweetEngagementEventPairSqlPath,

userTweetEngagementEventPairTemplateVariable

),

// Min interaction count filter

"MIN\_INTERACTION\_COUNT" -> minInteractionCount.toString,

// Min fav count filter

"TWEET\_INTERACTION\_WITH\_FAV\_COUNT\_FILTER\_SQL" -> getTweetInteractionTableWithFavCountFilter(

startTime,

endTime,

minFavCount

),

// Health filter

"TWEET\_INTERACTION\_WITH\_HEALTH\_FILTER\_SQL" -> getTweetInteractionTableWithHealthFilter(

startTime,

endTime,

enableHealthAndVideoFilters),

"CONSUMER\_EMBEDDINGS\_SQL" -> consumerEmbeddingsSQL,

"TWEET\_EMBEDDING\_LENGTH" -> tweetEmbeddingsLength.toString,

"MIN\_ENGAGEMENT\_PER\_CLUSTER" -> minEngagementPerCluster.toString,

"CLUSTER\_TOP\_K\_TWEETS" -> clusterTopKTweets.toString

)

val query = BQQueryUtils.getBQQueryFromSqlFile(

simclustersEngagementBasedIndexGenerationSQLPath,

indexGenerationTemplateVariables)

val postFilterQuery = if (enableFavClusterTopKTweetsIntersection) {

generateClusterTopTweetIntersectionWithFavBasedIndexSQL(

startTime,

endTime,

clusterTopKTweets,

query)

} else {

query

}

// Generate SimClusters cluster-to-tweet index

sc.customInput(

s"SimClusters cluster-to-tweet index generation BQ job",

BigQueryIO

.read(parseClusterTopKTweetsFn(tweetEmbeddingsHalfLife))

.fromQuery(postFilterQuery)

.usingStandardSql()

)

}

}