package com.twitter.simclusters\_v2.scio

package bq\_generation.common

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

import org.joda.time.DateTime

object BQGenerationUtil {

// Consumer Embeddings BQ table details

val interestedInEmbeddings20M145K2020Table = BQTableDetails(

"twttr-bq-cassowary-prod",

"user",

"simclusters\_v2\_user\_to\_interested\_in\_20M\_145K\_2020",

)

val mtsConsumerEmbeddingsFav90P20MTable = BQTableDetails(

"twttr-bq-cassowary-prod",

"user",

"mts\_consumer\_embeddings\_fav90p\_20m",

)

// Common SQL path

val TweetFavCountSQLPath =

s"/com/twitter/simclusters\_v2/scio/bq\_generation/sql/tweet\_fav\_count.sql"

val NSFWTweetIdDenylistSQLPath =

s"/com/twitter/simclusters\_v2/scio/bq\_generation/sql/nsfw\_tweet\_denylist.sql"

val ClusterTopTweetsIntersectionWithFavBasedIndexSQLPath =

s"/com/twitter/simclusters\_v2/scio/bq\_generation/sql/cluster\_top\_tweets\_intersection\_with\_fav\_based\_index.sql"

// Read InterestedIn 2020

def getInterestedIn2020SQL(

queryDate: DateTime,

lookBackDays: Int

): String = {

s"""

|SELECT userId,

| clusterIdToScores.key AS clusterId,

| clusterIdToScores.value.logFavScore AS userScore,

| clusterIdToScores.value.logFavScoreClusterNormalizedOnly AS clusterNormalizedLogFavScore,

|FROM `$interestedInEmbeddings20M145K2020Table`, UNNEST(clusterIdToScores) AS clusterIdToScores

|WHERE DATE(\_PARTITIONTIME) =

| ( -- Get latest partition time

| SELECT MAX(DATE(\_PARTITIONTIME)) latest\_partition

| FROM `$interestedInEmbeddings20M145K2020Table`

| WHERE Date(\_PARTITIONTIME) BETWEEN

| DATE\_SUB(Date("${queryDate}"),

| INTERVAL $lookBackDays DAY) AND DATE("$queryDate")

| )

| AND clusterIdToScores.value.logFavScore > 0.0 # min score threshold for user embedding values

|""".stripMargin

}

// Read MTS Consumer Embeddings - Fav90P20M config

def getMTSConsumerEmbeddingsFav90P20MSQL(

queryDate: DateTime,

lookBackDays: Int

): String = {

// We read the most recent snapshot of MTS Consumer Embeddings Fav90P20M

s"""

|SELECT userId,

| clusterIdToScores.key AS clusterId,

| clusterIdToScores.value.logFavUserScore AS userScore,

| clusterIdToScores.value.logFavUserScoreClusterNormalized AS clusterNormalizedLogFavScore

| FROM `$mtsConsumerEmbeddingsFav90P20MTable`, UNNEST(embedding.clusterIdToScores) AS clusterIdToScores

|WHERE DATE(ingestionTime) = (

| -- Get latest partition time

| SELECT MAX(DATE(ingestionTime)) latest\_partition

| FROM `$mtsConsumerEmbeddingsFav90P20MTable`

| WHERE Date(ingestionTime) BETWEEN

| DATE\_SUB(Date("${queryDate}"),

| INTERVAL $lookBackDays DAY) AND DATE("${queryDate}")

|) AND clusterIdToScores.value.logFavUserScore > 0.0

|""".stripMargin

}

/\*

\* For a specific tweet engagement, retrieve the user id, tweet id, and timestamp

\*

\* Return:

\* String - UserId, TweetId and Timestamp table SQL string format

\* Table Schema

\* - userId: Long

\* - tweetId: Long

\* - tsMillis: Long

\*/

def getUserTweetEngagementEventPairSQL(

startTime: DateTime,

endTime: DateTime,

userTweetEngagementEventPairSqlPath: String,

userTweetEngagementEventPairTemplateVariable: Map[String, String]

): String = {

val templateVariables = Map(

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

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

"NO\_OLDER\_TWEETS\_THAN\_DATE" -> startTime.toString()

) ++ userTweetEngagementEventPairTemplateVariable

BQQueryUtils.getBQQueryFromSqlFile(userTweetEngagementEventPairSqlPath, templateVariables)

}

/\*

\* Retrieve tweets and the # of favs it got from a given time window

\*

\* Return:

\* String - TweetId and fav count table SQL string format

\* Table Schema

\* - tweetId: Long

\* - favCount: Long

\*/

def getTweetIdWithFavCountSQL(

startTime: DateTime,

endTime: DateTime,

): String = {

val templateVariables =

Map(

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

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

)

BQQueryUtils.getBQQueryFromSqlFile(TweetFavCountSQLPath, templateVariables)

}

/\*

\* From a given time window, retrieve tweetIds that were created by specific author or media type

\*

\* Input:

\* - startTime: DateTime

\* - endTime: DateTime

\* - filterMediaType: Option[Int]

\* MediaType

\* 1: Image

\* 2: GIF

\* 3: Video

\* - filterNSFWAuthor: Boolean

\* Whether we want to filter out NSFW tweet authors

\*

\* Return:

\* String - TweetId table SQL string format

\* Table Schema

\* - tweetId: Long

\*/

def getTweetIdWithMediaAndNSFWAuthorFilterSQL(

startTime: DateTime,

endTime: DateTime,

filterMediaType: Option[Int],

filterNSFWAuthor: Boolean

): String = {

val sql = s"""

|SELECT DISTINCT tweetId

|FROM `twttr-bq-tweetsource-prod.user.unhydrated\_flat` tweetsource, UNNEST(media) AS media

|WHERE (DATE(\_PARTITIONTIME) >= DATE("${startTime}") AND DATE(\_PARTITIONTIME) <= DATE("${endTime}")) AND

| timestamp\_millis((1288834974657 +

| ((tweetId & 9223372036850581504) >> 22))) >= TIMESTAMP("${startTime}")

| AND timestamp\_millis((1288834974657 +

| ((tweetId & 9223372036850581504) >> 22))) <= TIMESTAMP("${endTime}")

|""".stripMargin

val filterMediaStr = filterMediaType match {

case Some(mediaType) => s" AND media.media\_type =${mediaType}"

case \_ => ""

}

val filterNSFWAuthorStr = if (filterNSFWAuthor) " AND nsfwUser = false" else ""

sql + filterMediaStr + filterNSFWAuthorStr

}

/\*

\* From a given time window, retrieve tweetIds that fall into the NSFW deny list

\*

\* Input:

\* - startTime: DateTime

\* - endTime: DateTime

\*

\* Return:

\* String - TweetId table SQL string format

\* Table Schema

\* - tweetId: Long

\*/

def getNSFWTweetIdDenylistSQL(

startTime: DateTime,

endTime: DateTime,

): String = {

val templateVariables =

Map(

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

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

)

BQQueryUtils.getBQQueryFromSqlFile(NSFWTweetIdDenylistSQLPath, templateVariables)

}

/\*

\* From a given cluster id to top k tweets table and a time window,

\* (1) Retrieve the latest fav-based top tweets per cluster table within the time window

\* (2) Inner join with the given table using cluster id and tweet id

\* (3) Create the top k tweets per cluster table for the intersection

\*

\* Input:

\* - startTime: DateTime

\* - endTime: DateTime

\* - topKTweetsForClusterKeySQL: String, a SQL query

\*

\* Return:

\* String - TopKTweetsForClusterKey table SQL string format

\* Table Schema

\* - clusterId: Long

\* - topKTweetsForClusterKey: (Long, Long)

\* - tweetId: Long

\* - tweetScore: Long

\*/

def generateClusterTopTweetIntersectionWithFavBasedIndexSQL(

startTime: DateTime,

endTime: DateTime,

clusterTopKTweets: Int,

topKTweetsForClusterKeySQL: String

): String = {

val templateVariables =

Map(

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

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

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

"CLUSTER\_TOP\_TWEETS\_SQL" -> topKTweetsForClusterKeySQL

)

BQQueryUtils.getBQQueryFromSqlFile(

ClusterTopTweetsIntersectionWithFavBasedIndexSQLPath,

templateVariables)

}

/\*

\* Given a list of action types, build a string that indicates the user

\* engaged with the tweet

\*

\* Example use case: We want to build a SQL query that specifies this user engaged

\* with tweet with either fav or retweet actions.

\*

\* Input:

\* - actionTypes: Seq("ServerTweetFav", "ServerTweetRetweet")

\* - booleanOperator: "OR"

\* Output: "ServerTweetFav.engaged = 1 OR ServerTweetRetweet.engaged = 1"

\*

\* Example SQL:

\* SELECT ServerTweetFav, ServerTweetRetweet

\* FROM table

\* WHERE ServerTweetFav.engaged = 1 OR ServerTweetRetweet.engaged = 1

\*/

def buildActionTypesEngagementIndicatorString(

actionTypes: Seq[String],

booleanOperator: String = "OR"

): String = {

actionTypes.map(action => f"""${action}.engaged = 1""").mkString(f""" ${booleanOperator} """)

}

}

case class BQTableDetails(

projectName: String,

tableName: String,

datasetName: String) {

override def toString: String = s"${projectName}.${tableName}.${datasetName}"

}