package com.twitter.interaction\_graph.scio.ml.scores

import com.google.cloud.bigquery.BigQueryOptions

import com.google.cloud.bigquery.QueryJobConfiguration

import com.spotify.scio.ScioContext

import com.spotify.scio.values.SCollection

import com.twitter.beam.io.dal.DAL

import com.twitter.beam.io.exception.DataNotFoundException

import com.twitter.beam.io.fs.multiformat.PathLayout

import com.twitter.scalding\_internal.multiformat.format.keyval.KeyVal

import com.twitter.scio\_internal.job.ScioBeamJob

import com.twitter.wtf.candidate.thriftscala.Candidate

import com.twitter.wtf.candidate.thriftscala.CandidateSeq

import com.twitter.wtf.candidate.thriftscala.ScoredEdge

import org.apache.avro.generic.GenericRecord

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

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

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

import org.apache.beam.sdk.transforms.SerializableFunction

import scala.collection.JavaConverters.\_

object InteractionGraphScoreExportJob extends ScioBeamJob[InteractionGraphScoreExportOption] {

// to parse latest date from the BQ table we're reading from

val parseDateRow = new SerializableFunction[SchemaAndRecord, String] {

override def apply(input: SchemaAndRecord): String = {

val genericRecord: GenericRecord = input.getRecord()

genericRecord.get("ds").toString

}

}

// to parse each row from the BQ table we're reading from

val parseRow = new SerializableFunction[SchemaAndRecord, ScoredEdge] {

override def apply(record: SchemaAndRecord): ScoredEdge = {

val genericRecord: GenericRecord = record.getRecord()

ScoredEdge(

genericRecord.get("source\_id").asInstanceOf[Long],

genericRecord.get("destination\_id").asInstanceOf[Long],

genericRecord.get("prob").asInstanceOf[Double],

genericRecord.get("followed").asInstanceOf[Boolean],

)

}

}

override def runPipeline(

sc: ScioContext,

opts: InteractionGraphScoreExportOption

): Unit = {

val dateStr: String = opts.getDate().value.getStart.toString("yyyyMMdd")

logger.info(s"dateStr $dateStr")

val project: String = "twttr-recos-ml-prod"

val datasetName: String = "realgraph"

val bqTableName: String = "scores"

val fullBqTableName: String = s"$project:$datasetName.$bqTableName"

if (opts.getDALWriteEnvironment == "PROD") {

val bqClient =

BigQueryOptions.newBuilder.setProjectId("twttr-recos-ml-prod").build.getService

val query =

s"""

|SELECT total\_rows

|FROM `$project.$datasetName.INFORMATION\_SCHEMA.PARTITIONS`

|WHERE partition\_id ="$dateStr" AND

|table\_name="$bqTableName" AND total\_rows > 0

|""".stripMargin

val queryConfig = QueryJobConfiguration.of(query)

val results = bqClient.query(queryConfig).getValues.asScala.toSeq

if (results.isEmpty || results.head.get(0).getLongValue == 0) {

throw new DataNotFoundException(s"$dateStr not present in $fullBqTableName.")

}

}

sc.run()

}

override protected def configurePipeline(

sc: ScioContext,

opts: InteractionGraphScoreExportOption

): Unit = {

val dateStr: String = opts.getDate().value.getStart.toString("yyyy-MM-dd")

logger.info(s"dateStr $dateStr")

val project: String = "twttr-recos-ml-prod"

val datasetName: String = "realgraph"

val bqTableName: String = "scores"

val fullBqTableName: String = s"$project:$datasetName.$bqTableName"

val scoreExport: SCollection[ScoredEdge] = sc

.customInput(

s"Read from BQ table $fullBqTableName",

BigQueryIO

.read(parseRow)

.from(fullBqTableName)

.withSelectedFields(List("source\_id", "destination\_id", "prob", "followed").asJava)

.withRowRestriction(s"ds = '$dateStr'")

.withMethod(TypedRead.Method.DIRECT\_READ)

)

val inScores = scoreExport

.collect {

case ScoredEdge(src, dest, score, true) =>

(src, Candidate(dest, score))

}

.groupByKey

.map {

case (src, candidateIter) => KeyVal(src, CandidateSeq(candidateIter.toSeq.sortBy(-\_.score)))

}

val outScores = scoreExport

.collect {

case ScoredEdge(src, dest, score, false) =>

(src, Candidate(dest, score))

}

.groupByKey

.map {

case (src, candidateIter) => KeyVal(src, CandidateSeq(candidateIter.toSeq.sortBy(-\_.score)))

}

inScores.saveAsCustomOutput(

"Write real\_graph\_in\_scores",

DAL.writeVersionedKeyVal(

RealGraphInScoresScalaDataset,

PathLayout.VersionedPath(opts.getOutputPath + "/in"),

)

)

outScores.saveAsCustomOutput(

"Write real\_graph\_oon\_scores",

DAL.writeVersionedKeyVal(

RealGraphOonScoresScalaDataset,

PathLayout.VersionedPath(opts.getOutputPath + "/oon"),

)

)

}

}