package com.twitter.follow\_recommendations.common.predicates.sgs

import com.google.common.annotations.VisibleForTesting

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.follow\_recommendations.common.base.Predicate

import com.twitter.follow\_recommendations.common.base.PredicateResult

import com.twitter.follow\_recommendations.common.models.CandidateUser

import com.twitter.follow\_recommendations.common.models.FilterReason.InvalidRelationshipTypes

import com.twitter.socialgraph.thriftscala.ExistsRequest

import com.twitter.socialgraph.thriftscala.ExistsResult

import com.twitter.socialgraph.thriftscala.LookupContext

import com.twitter.socialgraph.thriftscala.Relationship

import com.twitter.socialgraph.thriftscala.RelationshipType

import com.twitter.stitch.Stitch

import com.twitter.stitch.socialgraph.SocialGraph

import com.twitter.util.logging.Logging

import javax.inject.Inject

import javax.inject.Singleton

class SgsRelationshipsByUserIdPredicate(

socialGraph: SocialGraph,

relationshipMappings: Seq[RelationshipMapping],

statsReceiver: StatsReceiver)

extends Predicate[(Option[Long], CandidateUser)]

with Logging {

private val InvalidFromPrimaryCandidateSourceName = "invalid\_from\_primary\_candidate\_source"

private val InvalidFromCandidateSourceName = "invalid\_from\_candidate\_source"

private val NoPrimaryCandidateSource = "no\_primary\_candidate\_source"

private val stats: StatsReceiver = statsReceiver.scope(this.getClass.getName)

override def apply(

pair: (Option[Long], CandidateUser)

): Stitch[PredicateResult] = {

val (idOpt, candidate) = pair

val relationships = relationshipMappings.map { relationshipMapping: RelationshipMapping =>

Relationship(

relationshipMapping.relationshipType,

relationshipMapping.includeBasedOnRelationship)

}

idOpt

.map { id: Long =>

val existsRequest = ExistsRequest(

id,

candidate.id,

relationships = relationships,

context = SgsRelationshipsByUserIdPredicate.UnionLookupContext

)

socialGraph

.exists(existsRequest).map { existsResult: ExistsResult =>

if (existsResult.exists) {

candidate.getPrimaryCandidateSource match {

case Some(candidateSource) =>

stats

.scope(InvalidFromPrimaryCandidateSourceName).counter(

candidateSource.name).incr()

case None =>

stats

.scope(InvalidFromPrimaryCandidateSourceName).counter(

NoPrimaryCandidateSource).incr()

}

candidate.getCandidateSources.foreach({

case (candidateSource, \_) =>

stats

.scope(InvalidFromCandidateSourceName).counter(candidateSource.name).incr()

})

PredicateResult.Invalid(Set(InvalidRelationshipTypes(relationshipMappings

.map { relationshipMapping: RelationshipMapping =>

relationshipMapping.relationshipType

}.mkString(", "))))

} else {

PredicateResult.Valid

}

}

}

// if no user id is present, return true by default

.getOrElse(Stitch.value(PredicateResult.Valid))

}

}

object SgsRelationshipsByUserIdPredicate {

// OR Operation

@VisibleForTesting

private[follow\_recommendations] val UnionLookupContext = Some(

LookupContext(performUnion = Some(true)))

}

@Singleton

class ExcludeNonFollowersSgsPredicate @Inject() (

socialGraph: SocialGraph,

statsReceiver: StatsReceiver)

extends SgsRelationshipsByUserIdPredicate(

socialGraph,

Seq(RelationshipMapping(RelationshipType.FollowedBy, includeBasedOnRelationship = false)),

statsReceiver)

@Singleton

class ExcludeNonFollowingSgsPredicate @Inject() (

socialGraph: SocialGraph,

statsReceiver: StatsReceiver)

extends SgsRelationshipsByUserIdPredicate(

socialGraph,

Seq(RelationshipMapping(RelationshipType.Following, includeBasedOnRelationship = false)),

statsReceiver)

@Singleton

class ExcludeFollowingSgsPredicate @Inject() (

socialGraph: SocialGraph,

statsReceiver: StatsReceiver)

extends SgsRelationshipsByUserIdPredicate(

socialGraph,

Seq(RelationshipMapping(RelationshipType.Following, includeBasedOnRelationship = true)),

statsReceiver)