package com.twitter.frigate.pushservice.predicate.magic\_fanout

import com.twitter.datatools.entityservice.entities.sports.thriftscala.NflFootballGameLiveUpdate

import com.twitter.datatools.entityservice.entities.sports.thriftscala.SoccerMatchLiveUpdate

import com.twitter.datatools.entityservice.entities.sports.thriftscala.SoccerPeriod

import com.twitter.datatools.entityservice.entities.sports.thriftscala.SportsEventHomeAwayTeamScore

import com.twitter.datatools.entityservice.entities.sports.thriftscala.SportsEventStatus

import com.twitter.datatools.entityservice.entities.sports.thriftscala.SportsEventTeamAlignment.Away

import com.twitter.datatools.entityservice.entities.sports.thriftscala.SportsEventTeamAlignment.Home

import com.twitter.escherbird.metadata.thriftscala.EntityMegadata

import com.twitter.frigate.pushservice.params.SportGameEnum

import com.twitter.frigate.common.base.GenericGameScore

import com.twitter.frigate.common.base.NflGameScore

import com.twitter.frigate.common.base.SoccerGameScore

import com.twitter.frigate.common.base.TeamInfo

import com.twitter.frigate.common.base.TeamScore

import com.twitter.hermit.store.semantic\_core.SemanticEntityForQuery

import com.twitter.storehaus.ReadableStore

import com.twitter.util.Future

object MagicFanoutSportsUtil {

def transformSoccerGameScore(game: SoccerMatchLiveUpdate): Option[SoccerGameScore] = {

require(game.status.isDefined)

val gameScore = transformToGameScore(game.score, game.status.get)

val \_penaltyKicks = transformToGameScore(game.penaltyScore, game.status.get)

gameScore.map { score =>

val \_isGameEnd = game.status.get match {

case SportsEventStatus.Completed(\_) => true

case \_ => false

}

val \_isHalfTime = game.period.exists { period =>

period match {

case SoccerPeriod.Halftime(\_) => true

case \_ => false

}

}

val \_isOvertime = game.period.exists { period =>

period match {

case SoccerPeriod.PreOvertime(\_) => true

case \_ => false

}

}

val \_isPenaltyKicks = game.period.exists { period =>

period match {

case SoccerPeriod.PrePenalty(\_) => true

case SoccerPeriod.Penalty(\_) => true

case \_ => false

}

}

val \_gameMinute = game.gameMinute.map { soccerGameMinute =>

game.minutesInInjuryTime match {

case Some(injuryTime) => s"($soccerGameMinute+$injuryTime′)"

case None => s"($soccerGameMinute′)"

}

}

SoccerGameScore(

score.home,

score.away,

isGameOngoing = score.isGameOngoing,

penaltyKicks = \_penaltyKicks,

gameMinute = \_gameMinute,

isHalfTime = \_isHalfTime,

isOvertime = \_isOvertime,

isPenaltyKicks = \_isPenaltyKicks,

isGameEnd = \_isGameEnd

)

}

}

def transformNFLGameScore(game: NflFootballGameLiveUpdate): Option[NflGameScore] = {

require(game.status.isDefined)

val gameScore = transformToGameScore(game.score, game.status.get)

gameScore.map { score =>

val \_isGameEnd = game.status.get match {

case SportsEventStatus.Completed(\_) => true

case \_ => false

}

val \_matchTime = (game.quarter, game.remainingSecondsInQuarter) match {

case (Some(quarter), Some(remainingSeconds)) if remainingSeconds != 0L =>

val m = (remainingSeconds / 60) % 60

val s = remainingSeconds % 60

val formattedSeconds = "%02d:%02d".format(m, s)

s"(Q$quarter - $formattedSeconds)"

case (Some(quarter), None) => s"(Q$quarter)"

case \_ => ""

}

NflGameScore(

score.home,

score.away,

isGameOngoing = score.isGameOngoing,

isGameEnd = \_isGameEnd,

matchTime = \_matchTime

)

}

}

/\*\*

Takes a score from Strato columns and turns it into an easier to handle structure (GameScore class)

We do this to easily access the home/away scenario for copy setting

\*/

def transformToGameScore(

scoreOpt: Option[SportsEventHomeAwayTeamScore],

status: SportsEventStatus

): Option[GenericGameScore] = {

val isGameOngoing = status match {

case SportsEventStatus.InProgress(\_) => true

case SportsEventStatus.Completed(\_) => false

case \_ => false

}

val scoresWithTeam = scoreOpt

.map { score =>

score.scores.map { score => (score.score, score.participantAlignment, score.participantId) }

}.getOrElse(Seq())

val tuple = scoresWithTeam match {

case Seq(teamOne, teamTwo, \_\*) => Some((teamOne, teamTwo))

case \_ => None

}

tuple.flatMap {

case ((Some(teamOneScore), teamOneAlignment, teamOne), (Some(teamTwoScore), \_, teamTwo)) =>

teamOneAlignment.flatMap {

case Home(\_) =>

val home = TeamScore(teamOneScore, teamOne.entityId, teamOne.domainId)

val away = TeamScore(teamTwoScore, teamTwo.entityId, teamTwo.domainId)

Some(GenericGameScore(home, away, isGameOngoing))

case Away(\_) =>

val away = TeamScore(teamOneScore, teamOne.entityId, teamOne.domainId)

val home = TeamScore(teamTwoScore, teamTwo.entityId, teamTwo.domainId)

Some(GenericGameScore(home, away, isGameOngoing))

case \_ => None

}

case \_ => None

}

}

def getTeamInfo(

team: TeamScore,

semanticCoreMegadataStore: ReadableStore[SemanticEntityForQuery, EntityMegadata]

): Future[Option[TeamInfo]] = {

semanticCoreMegadataStore

.get(SemanticEntityForQuery(team.teamDomainId, team.teamEntityId)).map {

\_.flatMap {

\_.basicMetadata.map { metadata =>

TeamInfo(

name = metadata.name,

twitterUserId = metadata.twitter.flatMap(\_.preferredTwitterUserId))

}

}

}

}

def getNFLReadableName(name: String): String = {

val teamNames =

Seq("")

teamNames.find(teamName => name.contains(teamName)).getOrElse(name)

}

def getSoccerIbisMap(game: SoccerGameScore): Map[String, String] = {

val gameMinuteMap = game.gameMinute

.map { gameMinute => Map("match\_time" -> gameMinute) }

.getOrElse(Map.empty)

val updateTypeMap = {

if (game.isGameEnd) Map("is\_game\_end" -> "true")

else if (game.isHalfTime) Map("is\_half\_time" -> "true")

else if (game.isOvertime) Map("is\_overtime" -> "true")

else if (game.isPenaltyKicks) Map("is\_penalty\_kicks" -> "true")

else Map("is\_score\_update" -> "true")

}

val awayScore = game match {

case SoccerGameScore(\_, away, \_, None, \_, \_, \_, \_, \_) =>

away.score.toString

case SoccerGameScore(\_, away, \_, Some(penaltyKick), \_, \_, \_, \_, \_) =>

s"${away.score} (${penaltyKick.away.score}) "

case \_ => ""

}

val homeScore = game match {

case SoccerGameScore(home, \_, \_, None, \_, \_, \_, \_, \_) =>

home.score.toString

case SoccerGameScore(home, \_, \_, Some(penaltyKick), \_, \_, \_, \_, \_) =>

s"${home.score} (${penaltyKick.home.score}) "

case \_ => ""

}

val scoresMap = Map(

"away\_score" -> awayScore,

"home\_score" -> homeScore,

)

gameType(SportGameEnum.Soccer) ++ updateTypeMap ++ gameMinuteMap ++ scoresMap

}

def getNflIbisMap(game: NflGameScore): Map[String, String] = {

val gameMinuteMap = Map("match\_time" -> game.matchTime)

val updateTypeMap = {

if (game.isGameEnd) Map("is\_game\_end" -> "true")

else Map("is\_score\_update" -> "true")

}

val awayScore = game.away.score

val homeScore = game.home.score

val scoresMap = Map(

"away\_score" -> awayScore.toString,

"home\_score" -> homeScore.toString,

)

gameType(SportGameEnum.Nfl) ++ updateTypeMap ++ gameMinuteMap ++ scoresMap

}

private def gameType(game: SportGameEnum.Value): Map[String, String] = {

game match {

case SportGameEnum.Soccer => Map("is\_soccer\_game" -> "true")

case SportGameEnum.Nfl => Map("is\_nfl\_game" -> "true")

case \_ => Map.empty

}

}

}