package com.twitter.tweetypie

package repository

import com.twitter.spam.rtf.thriftscala.SafetyLevel

import java.nio.ByteBuffer

object TweetQuery {

/\*\*

\* Parent trait that indicates what triggered the tweet query.

\*/

sealed trait Cause {

import Cause.\_

/\*\*

\* Is the tweet query hydrating the specified tweet for the purposes of a write?

\*/

def writing(tweetId: TweetId): Boolean =

this match {

case w: Write if w.tweetId == tweetId => true

case \_ => false

}

/\*\*

\* Is the tweet query performing a regular read for any tweet? If the cause is

\* a write on a different tweet, then any other tweet that is read in support of the write

\* is considered a normal read, and is subject to read-path hydration.

\*/

def reading(tweetId: TweetId): Boolean =

!writing(tweetId)

/\*\*

\* Are we performing an insert after create on the specified tweet? An undelete operation

\* performs an insert, but is not considered an initial insert.

\*/

def initialInsert(tweetId: TweetId): Boolean =

this match {

case Insert(`tweetId`) => true

case \_ => false

}

}

object Cause {

case object Read extends Cause

trait Write extends Cause {

val tweetId: TweetId

}

case class Insert(tweetId: TweetId) extends Write

case class Undelete(tweetId: TweetId) extends Write

}

/\*\*

\* Options for TweetQuery.

\*

\* @param include indicates which optionally hydrated fields on each tweet should be

\* hydrated and included.

\* @param enforceVisibilityFiltering whether Tweetypie visibility hydrators should be run to

\* filter protected tweets, blocked quote tweets, contributor data, etc. This does not affect

\* Visibility Library (http://go/vf) based filtering.

\* @param cause indicates what triggered the read: a normal read, or a write operation.

\* @param forExternalConsumption when true, the tweet is being read for rendering to an external

\* client such as the iPhone Twitter app and is subject to being Dropped to prevent serving

\* "bad" text to clients that might crash their OS. When false, the tweet is being read for internal

\* non-client purposes and should never be Dropped.

\* @param isInnerQuotedTweet Set by [[com.twitter.tweetypie.hydrator.QuotedTweetHydrator]],

\* to be used by [[com.twitter.visibility.interfaces.tweets.TweetVisibilityLibrary]]

\* so VisibilityFiltering library can execute Interstitial logic on inner quoted tweets.

\* @param fetchStoredTweets Set by GetStoredTweetsHandler. If set to true, the Manhattan storage

\* layer will fetch and construct Tweets regardless of what state they're in.

\*/

case class Options(

include: TweetQuery.Include,

cacheControl: CacheControl = CacheControl.ReadWriteCache,

cardsPlatformKey: Option[String] = None,

excludeReported: Boolean = false,

enforceVisibilityFiltering: Boolean = false,

safetyLevel: SafetyLevel = SafetyLevel.FilterNone,

forUserId: Option[UserId] = None,

languageTag: String = "en",

extensionsArgs: Option[ByteBuffer] = None,

cause: Cause = Cause.Read,

scrubUnrequestedFields: Boolean = true,

requireSourceTweet: Boolean = true,

forExternalConsumption: Boolean = false,

simpleQuotedTweet: Boolean = false,

isInnerQuotedTweet: Boolean = false,

fetchStoredTweets: Boolean = false,

isSourceTweet: Boolean = false,

enableEditControlHydration: Boolean = true)

case class Include(

tweetFields: Set[FieldId] = Set.empty,

countsFields: Set[FieldId] = Set.empty,

mediaFields: Set[FieldId] = Set.empty,

quotedTweet: Boolean = false,

pastedMedia: Boolean = false) {

/\*\*

\* Accumulates additional (rather than replaces) field ids.

\*/

def also(

tweetFields: Traversable[FieldId] = Nil,

countsFields: Traversable[FieldId] = Nil,

mediaFields: Traversable[FieldId] = Nil,

quotedTweet: Option[Boolean] = None,

pastedMedia: Option[Boolean] = None

): Include =

copy(

tweetFields = this.tweetFields ++ tweetFields,

countsFields = this.countsFields ++ countsFields,

mediaFields = this.mediaFields ++ mediaFields,

quotedTweet = quotedTweet.getOrElse(this.quotedTweet),

pastedMedia = pastedMedia.getOrElse(this.pastedMedia)

)

/\*\*

\* Removes field ids.

\*/

def exclude(

tweetFields: Traversable[FieldId] = Nil,

countsFields: Traversable[FieldId] = Nil,

mediaFields: Traversable[FieldId] = Nil

): Include =

copy(

tweetFields = this.tweetFields -- tweetFields,

countsFields = this.countsFields -- countsFields,

mediaFields = this.mediaFields -- mediaFields

)

def ++(that: Include): Include =

copy(

tweetFields = this.tweetFields ++ that.tweetFields,

countsFields = this.countsFields ++ that.countsFields,

mediaFields = this.mediaFields ++ that.mediaFields,

quotedTweet = this.quotedTweet || that.quotedTweet,

pastedMedia = this.pastedMedia || that.pastedMedia

)

}

}

sealed case class CacheControl(writeToCache: Boolean, readFromCache: Boolean)

object CacheControl {

val NoCache: CacheControl = CacheControl(false, false)

val ReadOnlyCache: CacheControl = CacheControl(false, true)

val ReadWriteCache: CacheControl = CacheControl(true, true)

}