package com.twitter.tweetypie.storage

/\*\*

\* Responsible for encoding/decoding Tweet records to/from Manhattan keys

\*

\* K/V Scheme:

\* -----------

\* [TweetId]

\* /metadata

\* /delete\_state (a.k.a. hard delete)

\* /soft\_delete\_state

\* /bounce\_delete\_state

\* /undelete\_state

\* /force\_added\_state

\* /scrubbed\_fields/

\* /[ScrubbedFieldId\_1]

\* ..

\* /[ScrubbedFieldId\_M]

\* /fields

\* /internal

\* /1

\* /9

\* ..

\* /99

\* /external

\* /100

\* ..

\*

\* IMPORTANT NOTE:

\* 1) Field Ids 2 to 8 in Tweet thrift struct are considered "core fields" are 'packed' together

\* into a TFieldBlob and stored under field id 1 (i.e [DatasetName]/[TweetId]/fields/internal/1).

\* This is why we do not see keys from [DatasetName]/[TweetId]/fields/internal/2 to [DatasetName]/

\* [TweetId]/fields/internal/8)

\*

\* 2) Also, the tweet id (which is the field id 1 in Tweet thrift structure) is not explicitly stored

\* in Manhattan. There is no need to explicitly store it since it is a part of the Pkey

\*/

case class TweetKey(tweetId: TweetId, lKey: TweetKey.LKey) {

override def toString: String =

s"/${ManhattanOperations.PkeyInjection(tweetId)}/${ManhattanOperations.LkeyInjection(lKey)}"

}

object TweetKey {

// Manhattan uses lexicographical order for keys. To make sure lexicographical order matches the

// numerical order, we should pad both tweet id and field ids with leading zeros.

// Since tweet id is long and field id is a short, the max width of each can be obtained by doing

// Long.MaxValue.toString.length and Short.MaxValue.toString.length respectively

private val TweetIdFormatStr = s"%0${Long.MaxValue.toString.length}d"

private val FieldIdFormatStr = s"%0${Short.MaxValue.toString.length}d"

private[storage] def padTweetIdStr(tweetId: Long): String = TweetIdFormatStr.format(tweetId)

private[storage] def padFieldIdStr(fieldId: Short): String = FieldIdFormatStr.format(fieldId)

def coreFieldsKey(tweetId: TweetId): TweetKey = TweetKey(tweetId, LKey.CoreFieldsKey)

def hardDeletionStateKey(tweetId: TweetId): TweetKey =

TweetKey(tweetId, LKey.HardDeletionStateKey)

def softDeletionStateKey(tweetId: TweetId): TweetKey =

TweetKey(tweetId, LKey.SoftDeletionStateKey)

def bounceDeletionStateKey(tweetId: TweetId): TweetKey =

TweetKey(tweetId, LKey.BounceDeletionStateKey)

def unDeletionStateKey(tweetId: TweetId): TweetKey = TweetKey(tweetId, LKey.UnDeletionStateKey)

def forceAddedStateKey(tweetId: TweetId): TweetKey = TweetKey(tweetId, LKey.ForceAddedStateKey)

def scrubbedGeoFieldKey(tweetId: TweetId): TweetKey = TweetKey(tweetId, LKey.ScrubbedGeoFieldKey)

def fieldKey(tweetId: TweetId, fieldId: FieldId): TweetKey =

TweetKey(tweetId, LKey.FieldKey(fieldId))

def internalFieldsKey(tweetId: TweetId, fieldId: FieldId): TweetKey =

TweetKey(tweetId, LKey.InternalFieldsKey(fieldId))

def additionalFieldsKey(tweetId: TweetId, fieldId: FieldId): TweetKey =

TweetKey(tweetId, LKey.AdditionalFieldsKey(fieldId))

def scrubbedFieldKey(tweetId: TweetId, fieldId: FieldId): TweetKey =

TweetKey(tweetId, LKey.ScrubbedFieldKey(fieldId))

// AllFieldsKeyPrefix: fields

// CoreFieldsKey: fields/internal/1 (Stores subset of StoredTweet fields which are

// "packed" into a single CoreFields record)

// HardDeletionStateKey: metadata/delete\_state

// SoftDeletionStateKey: metadata/soft\_delete\_state

// BounceDeletionStateKey: metadata/bounce\_delete\_state

// UnDeletionStateKey: metadata/undelete\_state

// ForceAddedStateKey: metadata/force\_added\_state

// FieldKey: fields/<group\_name>/<padded\_field\_id> (where <group\_name>

// is 'internal' for field ids < 100 and 'external' for all other

// fields ids)

// InternalFieldsKeyPrefix: fields/internal

// PKey: <empty string>

// ScrubbedFieldKey: metadata/scrubbed\_fields/<padded\_field\_id>

// ScrubbedFieldKeyPrefix: metadata/scrubbed\_fields

sealed abstract class LKey(override val toString: String)

object LKey {

private val HardDeletionRecordLiteral = "delete\_state"

private val SoftDeletionRecordLiteral = "soft\_delete\_state"

private val BounceDeletionRecordLiteral = "bounce\_delete\_state"

private val UnDeletionRecordLiteral = "undelete\_state"

private val ForceAddRecordLiteral = "force\_added\_state"

private val ScrubbedFieldsGroup = "scrubbed\_fields"

private val InternalFieldsGroup = "internal"

private val ExternalFieldsGroup = "external"

private val MetadataCategory = "metadata"

private val FieldsCategory = "fields"

private val InternalFieldsKeyPrefix = s"$FieldsCategory/$InternalFieldsGroup/"

private val ExternalFieldsKeyPrefix = s"$FieldsCategory/$ExternalFieldsGroup/"

private val ScrubbedFieldsKeyPrefix = s"$MetadataCategory/$ScrubbedFieldsGroup/"

sealed abstract class MetadataKey(metadataType: String)

extends LKey(s"$MetadataCategory/$metadataType")

sealed abstract class StateKey(stateType: String) extends MetadataKey(stateType)

case object HardDeletionStateKey extends StateKey(s"$HardDeletionRecordLiteral")

case object SoftDeletionStateKey extends StateKey(s"$SoftDeletionRecordLiteral")

case object BounceDeletionStateKey extends StateKey(s"$BounceDeletionRecordLiteral")

case object UnDeletionStateKey extends StateKey(s"$UnDeletionRecordLiteral")

case object ForceAddedStateKey extends StateKey(s"$ForceAddRecordLiteral")

case class ScrubbedFieldKey(fieldId: FieldId)

extends MetadataKey(s"$ScrubbedFieldsGroup/${padFieldIdStr(fieldId)}")

val ScrubbedGeoFieldKey: LKey.ScrubbedFieldKey = ScrubbedFieldKey(TweetFields.geoFieldId)

/\*\*

\* LKey that has one of many possible fields id. This generalize over

\* internal and additional fields key.

\*/

sealed abstract class FieldKey(prefix: String) extends LKey(toString) {

def fieldId: FieldId

override val toString: String = prefix + padFieldIdStr(fieldId)

}

object FieldKey {

def apply(fieldId: FieldId): FieldKey =

fieldId match {

case id if id < TweetFields.firstAdditionalFieldId => InternalFieldsKey(fieldId)

case \_ => AdditionalFieldsKey(fieldId)

}

}

case class InternalFieldsKey(fieldId: FieldId) extends FieldKey(InternalFieldsKeyPrefix) {

assert(fieldId < TweetFields.firstAdditionalFieldId)

}

case class AdditionalFieldsKey(fieldId: FieldId) extends FieldKey(ExternalFieldsKeyPrefix) {

assert(fieldId >= TweetFields.firstAdditionalFieldId)

}

val CoreFieldsKey: LKey.InternalFieldsKey = InternalFieldsKey(TweetFields.rootCoreFieldId)

case class Unknown private (str: String) extends LKey(str)

def fromString(str: String): LKey = {

def extractFieldId(prefix: String): FieldId =

str.slice(prefix.length, str.length).toShort

str match {

case CoreFieldsKey.toString => CoreFieldsKey

case HardDeletionStateKey.toString => HardDeletionStateKey

case SoftDeletionStateKey.toString => SoftDeletionStateKey

case BounceDeletionStateKey.toString => BounceDeletionStateKey

case UnDeletionStateKey.toString => UnDeletionStateKey

case ForceAddedStateKey.toString => ForceAddedStateKey

case ScrubbedGeoFieldKey.toString => ScrubbedGeoFieldKey

case \_ if str.startsWith(InternalFieldsKeyPrefix) =>

InternalFieldsKey(extractFieldId(InternalFieldsKeyPrefix))

case \_ if str.startsWith(ExternalFieldsKeyPrefix) =>

AdditionalFieldsKey(extractFieldId(ExternalFieldsKeyPrefix))

case \_ if str.startsWith(ScrubbedFieldsKeyPrefix) =>

ScrubbedFieldKey(extractFieldId(ScrubbedFieldsKeyPrefix))

case \_ => Unknown(str)

}

}

}

}