package com.twitter.servo.database

import com.twitter.util.Time

import java.sql.{ResultSet, Timestamp}

/\*\*

\* A base trait for transforming JDBC ResultSets.

\* Designed to be used with the Accessors trait.

\*/

trait ImplicitBuilder[T] extends Accessors {

def apply(implicit row: ResultSet): T

}

object Accessors {

/\*\*

\* helper to make it compile time error when trying to call getOption on types not supported

\* instead of a runtime exception

\*/

object SafeManifest {

implicit val booleanSafeManifest = new SafeManifest(implicitly[Manifest[Boolean]])

implicit val doubleSafeManifest = new SafeManifest(implicitly[Manifest[Double]])

implicit val intSafeManifest = new SafeManifest[Int](implicitly[Manifest[Int]])

implicit val longSafeManifest = new SafeManifest[Long](implicitly[Manifest[Long]])

implicit val stringSafeManifest = new SafeManifest[String](implicitly[Manifest[String]])

implicit val timestampSafeManifest =

new SafeManifest[Timestamp](implicitly[Manifest[Timestamp]])

}

@deprecated("safe manifests no longer supported, use type-specific accessors instead", "1.1.1")

case class SafeManifest[T](mf: Manifest[T])

}

/\*\*

\* mixin to get ResultSet accessors for standard types

\*/

trait Accessors {

import Accessors.\_

/\*\*

\* @return None when the column is null for the current row of the result set passed in

\* Some[T] otherwise

\* @throws UnsupportedOperationException if the return type expected is not supported, currently

\* only Boolean, Int, Long, String and Timestamp are supported

\*/

@deprecated("use type-specific accessors instead", "1.1.1")

def getOption[T](column: String)(implicit row: ResultSet, sf: SafeManifest[T]): Option[T] = {

val res = {

if (classOf[Boolean] == sf.mf.erasure) {

row.getBoolean(column)

} else if (classOf[Double] == sf.mf.erasure) {

row.getDouble(column)

} else if (classOf[Int] == sf.mf.erasure) {

row.getInt(column)

} else if (classOf[Long] == sf.mf.erasure) {

row.getLong(column)

} else if (classOf[String] == sf.mf.erasure) {

row.getString(column)

} else if (classOf[Timestamp] == sf.mf.erasure) {

row.getTimestamp(column)

} else {

throw new UnsupportedOperationException("type not supported: " + sf.mf.erasure)

}

}

if (row.wasNull()) {

None

} else {

Some(res.asInstanceOf[T])

}

}

/\*\*

\* @param get the method to apply to the ResultSet

\* @param row the implicit ResultSet on which to apply get

\* @return None when the column is null for the current row of the result set passed in

\* Some[T] otherwise

\*/

def getOption[T](get: ResultSet => T)(implicit row: ResultSet): Option[T] = {

val result = get(row)

if (row.wasNull()) {

None

} else {

Some(result)

}

}

def booleanOption(column: String)(implicit row: ResultSet): Option[Boolean] =

getOption((\_: ResultSet).getBoolean(column))

def boolean(column: String, default: Boolean = false)(implicit row: ResultSet): Boolean =

booleanOption(column).getOrElse(default)

def doubleOption(column: String)(implicit row: ResultSet): Option[Double] =

getOption((\_: ResultSet).getDouble(column))

def double(column: String, default: Double = 0.0)(implicit row: ResultSet): Double =

doubleOption(column).getOrElse(default)

def intOption(column: String)(implicit row: ResultSet): Option[Int] =

getOption((\_: ResultSet).getInt(column))

def int(column: String, default: Int = 0)(implicit row: ResultSet): Int =

intOption(column).getOrElse(default)

def longOption(column: String)(implicit row: ResultSet): Option[Long] =

getOption((\_: ResultSet).getLong(column))

def long(column: String, default: Long = 0)(implicit row: ResultSet): Long =

longOption(column).getOrElse(default)

def stringOption(column: String)(implicit row: ResultSet): Option[String] =

getOption((\_: ResultSet).getString(column))

def string(column: String, default: String = "")(implicit row: ResultSet): String =

stringOption(column).getOrElse(default)

def timestampOption(column: String)(implicit row: ResultSet): Option[Timestamp] =

getOption((\_: ResultSet).getTimestamp(column))

def timestamp(

column: String,

default: Timestamp = new Timestamp(0)

)(

implicit row: ResultSet

): Timestamp =

timestampOption(column).getOrElse(default)

def datetimeOption(column: String)(implicit row: ResultSet): Option[Long] =

timestampOption(column) map { \_.getTime }

def datetime(column: String, default: Long = 0L)(implicit row: ResultSet): Long =

datetimeOption(column).getOrElse(default)

def timeOption(column: String)(implicit row: ResultSet): Option[Time] =

datetimeOption(column) map { Time.fromMilliseconds(\_) }

def time(column: String, default: Time = Time.epoch)(implicit row: ResultSet): Time =

timeOption(column).getOrElse(default)

def bytesOption(column: String)(implicit row: ResultSet): Option[Array[Byte]] =

getOption((\_: ResultSet).getBytes(column))

def bytes(

column: String,

default: Array[Byte] = Array.empty[Byte]

)(

implicit row: ResultSet

): Array[Byte] =

bytesOption(column).getOrElse(default)

}