

memo-sqlite

Claude Heiland-Allen

2011-2018

Contents

1	Data/Memo/Sqlite.hs	2
2	examples/fib.hs	5
3	.gitignore	6
4	LICENSE	6
5	memo-sqlite.cabal	7
6	Setup.hs	7

1 Data/Memo/Sqlite.hs

```
{- |  
Module      : Data.Memo.Sqlite  
Copyright   : (c) 2011 Claude Heiland-Allen  
License     : BSD3
```

```
5  
Maintainer  : claud@mathr.co.uk  
Stability   : unstable  
Portability : portable
```

10 Memoize functions in a SQLite3 database.

The functions memoized while having type `@f :: k -> IO v@` must result in the same output given the same input, otherwise all kinds of wrongness will result.

15 The `@cleanup@` action returned by the memoizers must not be called if you are going to use the memoized function again; beware.

An example program (included in the examples directory) might look like:

```
20 > -- fib.hs  
> import Data.Memo.Sqlite (memoRec', readShow, table)  
>  
> import Control.Monad (liftM2)  
25 > import System.Environment (getArgs)  
>  
> fib :: (Integer -> IO Integer) -> Integer -> IO Integer  
> fib _fib ' n@0 = print n >> return 0  
> fib _fib ' n@1 = print n >> return 1  
30 > fib fib ' n = print n >> liftM2 (+) (fib ' (n - 1)) (fib ' (n - 2))  
>  
> main :: IO ()  
> main = do  
>   [file, ts, ns] <- getArgs  
35 >   let Just t = table ts
```

```

>     n = read ns
>     (cleanup, fib') <- memoRec' readShow file t fib
>     fib' n >>= \nth -> putStrLn $ "fib(" ++ show n ++ ") = " ++ show nth
>     cleanup

```

40

Example usage:

```

> ghc --make fib.hs
> ./fib fibs.sqlite3 fibs 10
45 > ./fib fibs.sqlite3 fibs 10
> ./fib fibs.sqlite3 fibs 100
> ./fib fibs.sqlite3 fibs 100

```

See also:

50

* the @data-memocombinators@ package for pure in-memory memoization.

-}

55 {-# LANGUAGE OverloadedStrings #-}

```

module Data.Memo.Sqlite
(
  -- * Database table identifier.
  60   Table()
  , table
  -- * Database (de)serialization.
  , Sqlite(..)
  -- ** Wrapper types for control over (de)serialization.
  65   , Wrap
  , Unwrap
  , Wrapper
  -- ** Read/Show (de)serialization.
  , ReadShow()
  , readShow
  70   -- * Memoizers.
  -- ** Memoizer types.
  , Memo
  , MemoRec
  75   , MkMemo
  , MkMemoRec
  -- ** Memoizer functions.
  , memo
  , memoRec
  80   , memo'
  , memoRec'
  -- * SQLite3 data (re-exported from the direct-sqlite package).
  , SQLData(..)
  ) where
85
import Prelude hiding (lookup, all)

import System.IO (fixIO)

90 import Data.Text as T

import Database.SQLite3 -- hackage: direct-sqlite

```

```

-- | A valid SQLite3 table name.
95 newtype Table = Table Text deriving (Eq, Ord, Show)

-- | Construct a table name.
table :: Text -> Maybe Table
table s
100   | T.null s = Nothing
   | T.head s 'elem' letters &&
     all ('elem' alphaNum) s &&
     not ("sqlite_" `isPrefixOf` s) = Just (Table $ "\"" < s < "\"")
   | otherwise = Nothing
105 where
     letters = ['a'..'z'] ++ ['A'..'Z']
     digits  = ['0'..'9']
     alphaNum = letters ++ digits ++ "_"

110 -- | Database (de)serialization
class Sqlite t where
  -- | Serialize to SQLite3 data.
  toSqlite :: t -> SQLData
  -- | Deserialize from SQLite3 data.
115 fromSqlite :: SQLData -> t

type Wrap    s t k v = (k -> IO v) -> s k -> IO (t v)
type Unwrap  s t k v = (s k -> IO (t v)) -> k -> IO v
type Wrapper s t k v = (Wrap s t k v, Unwrap s t k v)
120

-- | Use Read and Show for database (de)serialization.
newtype ReadShow t = ReadShow { unReadShow :: t }

instance (Read t, Show t) => Sqlite (ReadShow t) where
125   toSqlite (ReadShow t) = SQLText (T.pack $ show t)
   fromSqlite (SQLText s) = ReadShow $ read (T.unpack s)

toReadShow :: Wrap ReadShow ReadShow k v
toReadShow f k = ReadShow `fmap` f (unReadShow k)
130

fromReadShow :: Unwrap ReadShow ReadShow k v
fromReadShow f k = unReadShow `fmap` f (ReadShow k)

-- | Wrapper using Read and Show for (de)serialization of both keys and values.
135 readShow :: Wrapper ReadShow ReadShow k v
readShow = (toReadShow, fromReadShow)

type Memo k v = (k -> IO v) -> IO (IO (), k -> IO v)
type MemoRec k v = ((k -> IO v) -> k -> IO v) -> IO (IO (), k -> IO v)
140 type MkMemo k v = Text -> Table -> Memo k v
type MkMemoRec k v = Text -> Table -> MemoRec k v

-- | Memoize a function using an SQLite3 database.
memo :: (Sqlite k, Sqlite v) => MkMemo k v
145 memo file (Table tab) f = do
  db <- open file
  -- create database
  create <- prepare db $ "CREATE TABLE IF NOT EXISTS " < tab < " ( k TEXT ↵
    ↵ PRIMARY KEY, v TEXT )"

```

```

- <- step create
150 finalize create
-- prepare statements
lookup <- prepare db $ "SELECT v FROM " <> tab <> " WHERE k = ? LIMIT 1"
insert <- prepare db $ "INSERT INTO " <> tab <> " ( k , v ) VALUES ( ? , ? )"
let -- clean up database
155 cleanup = do
    finalize lookup
    finalize insert
    close db
-- memoize a value in the database
160 remember k = do
    -- lookup key
    let ks = toSqlite k
        reset lookup
        bind lookup [ks]
165 r <- step lookup
    case r of
        Row -> do -- found: return value
            vs <- column lookup 0
            return (fromSqlite vs)
170 Done -> do -- not found: calculate and insert
            v <- f k
            let vs = toSqlite v
                reset insert
                bind insert [ks, vs]
175 - <- step insert
            return v
    return ( cleanup , remember )

-- | Memoize a recursive function using an SQLite3 database.
180 memoRec :: (Sqlite k, Sqlite v) => MkMemoRec k v
memoRec file tab f = fixIO $ \rf -> memo file tab $ f (snd rf)

-- | Memoize a function using an SQLite3 database, using the supplied wrapper ↯
    ↵ for control of (de)serialization.
memo' :: (Sqlite (s k), Sqlite (t v)) => Wrapper s t k v -> MkMemo k v
185 memo' (wrap, unwrap) file tab = via wrap unwrap (memo file tab)

via i o a f = fmap o 'fmap' a (i f)

-- | Memoize a recursive function using an SQLite3 database, using the supplied ↯
    ↵ wrapper for control of (de)serialization.
190 memoRec' :: (Sqlite (s k), Sqlite (t v)) => Wrapper s t k v -> MkMemoRec k v
memoRec' (wrap, unwrap) file tab = viaRec wrap unwrap (memoRec file tab)

viaRec i o a f = fmap o 'fmap' a (i . f . o)

```

2 examples/fib.hs

```

import Control.Monad (liftM2)
import System.Environment (getArgs)
import qualified Data.Text as T
import Data.Memo.Sqlite (memoRec', readShow, table)
5
-- | Compute Fibonacci numbers.
fib :: (Integer -> IO Integer) -> Integer -> IO Integer

```

```

fib _fib ' n@0 = print n >> return 0
fib _fib ' n@1 = print n >> return 1
10 fib fib ' n = print n >> liftM2 (+) (fib ' (n - 1)) (fib ' (n - 2))

-- | Example program.
main :: IO ()
main = do
15   [file , ts , ns] <- getArgs
      let Just t = table (T.pack ts)
          n = read ns
          (cleanup , fib ') <- memoRec' readShow (T.pack file) t fib
          fib ' n >>= \fn -> putStrLn $ "fib(" ++ show n ++ ") = " ++ show fn
20   cleanup

{-
Example usage:
25   $ ghc -O2 -Wall --make fib.hs
      $ ./fib fibs.sqlite3 fibs 10
      $ ./fib fibs.sqlite3 fibs 10
      $ ./fib fibs.sqlite3 fibs 100
      $ ./fib fibs.sqlite3 fibs 100
30 -}

```

3 .gitignore

```

.cabal-sandbox
cabal.sandbox.config
dist

```

4 LICENSE

Copyright (c)2011, Claude Heiland-Allen

All rights reserved.

```

5 Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

    * Redistributions of source code must retain the above copyright
      notice, this list of conditions and the following disclaimer.
10    * Redistributions in binary form must reproduce the above
      copyright notice, this list of conditions and the following
      disclaimer in the documentation and/or other materials provided
      with the distribution.
15    * Neither the name of Claude Heiland-Allen nor the names of other
      contributors may be used to endorse or promote products derived
      from this software without specific prior written permission.

20 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
25 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT

```

LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 30 OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

5 memo-sqlite.cabal

```
Name: memo-sqlite
Version: 0.2
Synopsis: memoize functions using SQLite3 database
Homepage: https://code.mathr.co.uk/memo-sqlite
5 License: BSD3
License-file: LICENSE
Author: Claude Heiland-Allen
Maintainer: claud@mathr.co.uk
Category: Database
10 Build-type: Simple
Extra-source-files: examples/fib.hs
Cabal-version: >=1.10
```

Library

```
15 Exposed-modules: Data.Memo.Sqlite
Build-depends: base < 5,
direct-sqlite >= 2.3 && < 2.4,
text >= 1.2 && < 1.3
default-language: Haskell2010
20 other-extensions: OverloadedStrings
```

source-repository head

```
type: git
location: https://code.mathr.co.uk/memo-sqlite.git
25
```

source-repository this

```
type: git
location: https://code.mathr.co.uk/memo-sqlite.git
tag: memo-sqlite-0.2
```

6 Setup.hs

```
import Distribution.Simple
main = defaultMain
```