`easydb`, a simple database interface for SQLite, MS Access, MS Excel and MySQL

Julien Moeys

November 15, 2014

Contents

1 Forewords

1.1 What is `easydb`?

`easydb` is an R[2] package providing functions to easily read and write data from / to SQLite, MS Access, MS Excel and MySQL databases, and perform a few other operations. `easydb` provide the same interface (set of functions) for all these databases. It is build on top of `RSQLite`[1] and `RODBC`[3], but tries to hide tedious operations such as opening and closing database connections, or writing SQL queries. `easydb` provides S3 classes functions to manipulate databases in a similar way as a `data.frame` (single square brackets [ ] subsetting). Nonetheless it does not work exactly as a `data.frame` (because a `data.frame` have rows and columns, while a databases have tables, rows and columns).

1.2 Credits and License

`easydb` is licensed under an Affero GNU General Public License Version 3.

This package and this document is provided with NO responsibilities, guarantees or automatic supports from the author or his employer (SLU / CKB).

Many ideas behind `easydb` have been introduced before by **John Fox and Oswaldo Cruz** in their package `dfdb`, in a very similar form as in this package.

1.3 Disclaimer

The author of this package is neither a database/SQL specialist (rather a self taught database user), nor an R guRu. Some functions in `easydb` might be coded in a sub-optimal way. They should nonetheless work as expected. Please let me know if that is not the case!

2 Working with easydb

2.1 Install and load easydb

Method 1: If you have the latest R version, open R, and then type:

```
install.packages(
    pkgs  = c( "easydb", "easyrsqlite", "easyrodbcexcel",
        "easyrodbcaccess"),
    repos = "http://R-Forge.R-project.org" )
```

Method 2: Otherwise, try to install the package from the binaries. First download the binaries from `http://r-forge.r-project.org/R/?group_id=1200`. Save the package binaries in your working directory, and then open R and type:

```
install.packages(
    pkgs  = "easydb_0.3.1.zip", # add more package names
    repos = NULL )
```

Then you can load `easydb`:

```
library( "easydb" )
library( "easyrsqlite" )
# library( "easyrodbcexcel" )
# library( "easyrodbcaccess" )
# library( "easyrodbcmysql" )
```

2.2 SQLite and RSQLite

To work with SQLite databases, you need to install the R package RSQLite:

```
install.packages( "RSQLite" )
library( "RSQLite" )
```

2.3 ODBC and Access

If you want to use MS Access databases with `easydb`, you need a computer with ODBC (installation not covered by this manual), and the R package RODBC.

```
install.packages( "RODBC" )
library( "RODBC" )
```

2.4 ODBC and Excel

If you want to use MS Excel files with `easydb`, you need a computer with ODBC (installation not covered by this manual), and the R package RODBC.

```
install.packages( "RODBC" )
library( "RODBC" )
```

2.5 ODBC and MySQL

If you want to use MySQL with `easydb`, you need a computer with ODBC (installation not covered by this manual). You also need to install a MySQL driver for ODBC, that you can find here: `http://dev.mysql.com/downloads/connector/odbc/`. You also need the R package RODBC:

```
install.packages( "RODBC" )
library( "RODBC" )
```

2.6 Example databases

Some example databases are provided with the package. They are located in the folder:

```
system.file( package = "easydb" )
```
```
[1] "/home/rforge/lib/R/3.1/easydb"
```

Here is the list of files you can find in that folder:

```
list.files( system.file( package = "easydb" ) )
```
```
[1] "DESCRIPTION" "INDEX"       "Meta"        "NAMESPACE"
[5] "NEWS"        "R"           "help"        "html"
```

All the example databases are the same (but for diffferent database systems). They are pseudo-databases containing the physico-chemical properties of some soil profiles (but data are dummy).

At present, you can not create databases with `easydb`, but that feature should come sooner or later (except for MySQL).

2.6.1 SQLite

The example SQLite database is:

```
system.file( "soils.db", package = "easyrsqlite" )
```
```
[1] "/tmp/RtmpFLp1I3/Rinst42ea22501f97/easyrsqlite/soils.db"
```

2.6.2 MS Access (< 2007)

The example MS Access database is:

```
system.file( "soils.mdb", package = "easyrodbcaccess" )
```

2.6.3 MS Access (2007)

The example MS Access database is:

```
system.file( "soils.accdb", package = "easyrodbcaccess" )
```

2.6.4 MS Excel (< 2007)

The example MS Excel file is:

```
system.file( "soils.xls", package = "easyrodbcexcel" )
```

2.6.5 MS Excel (2007)

The example MS Excel file is:

```
system.file( "soils.xlsx", package = "easyrodbcexcel" )
```

2.6.6 MySQL

The example MySQL database can be created using the following SQL code file (dump that in a blank database, using PhPMyAdmin for example):

```
system.file( "create_MySQL_db.sql", package = "easyrodbcmysql" )
```

2.6.7 Copy the example database to the working directory

In order to avoid altering the original example database, it is better to copy them in the working directory. In this tutorial, we will work with the SQLite database only:

```
file.copy(
    from      = system.file( "soils.db", package = "easyrsqlite" ),
    to        = "soils.db",
    overwrite = TRUE )
```

```
[1] TRUE
```

```
#
# Check that the file has been copied:
file.exists( "soils.db" )
```

```
[1] TRUE
```

For an MS Access (< 2007) database you would do:

```
file.copy(
    from      = system.file( "soils.mdb", package = "easyrodbcaccess" ),
    to        = "soils.mdb",
    overwrite = TRUE )
```

For an MS Access 2007 database you would do:

```
file.copy(
    from      = system.file( "soils.accdb", package = "easyrodbcaccess" ),
    to        = "soils.accdb",
    overwrite = TRUE )
```

For an MS Excel (< 2007) file you would do:

```
file.copy(
    from      = system.file( "soils.xls", package = "easyrodbcexcel" ),
    to        = "soils.xls",
    overwrite = TRUE )
```

For an MS Excel 2007 file you would do:

```
file.copy(
    from      = system.file( "soils.xlsx", package = "easyrodbcexcel" ),
    to        = "soils.xlsx",
    overwrite = TRUE )
```

2.7 First step: 'describe' you database

The path and name of the database you are using is not enough for `easydb`, because (a) it is not possible to know from the file name which database type it is and (b) not all databases are described by a file path and name (MySQL for instance).

So you need to 'describe' a bit your database, using the function `edb()`. Notice that `edb()` does not create any connection to the database (so no need to close any connection after `edb()`). It just creates an R object containing the database description that will be recognised by R as an "`easydb` database" (with so called S3 classes).

This step is the **only** step that has database-specific requirements. The rest of the database handly will use the same set of functions, whatever the database system.

2.7.1 SQLite

To describe an SQLite database (here the example database), type:

```
sDb <- edb( dbName = "soils.db", dbType = "RSQLite_SQLite" )
```

`edb()` arguments are:

- `dbName`: Name and path of the database, as a text string;
- `dbType`: Database type. Here the string `RSQLite_SQLite` should be understood as "An SQLite database managed (internally) by the RSQLite package";

We now have an object called `sDb` that can be used for all our database operations.

```
# Object class:
class( sDb )
```

```
[1] "edb"            "RSQLite_SQLite"
```

2.7.2 MS Access (< 2007)

To describe an MS Access database (here the example database), type:

```
aDb <- edb( dbName = "soils.mdb", dbType = "RODBC_Access" )
```

`edb()` arguments are:

- `dbName`: Name and path of the database, as a text string;
- `dbType`: Database type. Here the string `RODBC_Access` should be understood as "An Access database managed (internally) by the RODBC package";

We now have an object called `aDb` that can be used for all our database operations.

```
# Object class:
class( aDb )
```

```
[1] "edb"          "RODBC_Access"
```

2.7.3 MS Access (2007)

To describe an MS Access 2007 database (here the example database), type:

```
aDb2 <- edb( dbName = "soils.accdb", dbType = "RODBC_Access",
    accessVersion = 2007 )
```

`edb()` arguments are:

- `dbName`: Name and path of the database, as a text string;
- `dbType`: Database type. Here the string `RODBC_Access` should be understood as "An Access database managed (internally) by the RODBC package";
- `accessVersion`: Single integer. Access version. Must be set to `2007` for MS Access 2007. Can be ignored for earlier versions of MS Access.

We now have an object called `aDb2` that can be used for all our database operations.

```
# Object class:
class( aDb2 )
```

```
[1] "edb"          "RODBC_Access"
```

2.7.4 MS Excel (< 2007)

To describe an MS Excel file (here the example database), type:

```
eDb <- edb( dbName = "soils.xls", dbType = "RODBC_Excel" )
```

`edb()` arguments are:

- `dbName`: Name and path of the database, as a text string;
- `dbType`: Database type. Here the string `RODBC_Excel` should be understood as "An Excel file managed (internally) by the RODBC package";

We now have an object called `eDb` that can be used for all our database operations.

```
# Object class:
class( eDb )
```

```
[1] "edb"         "RODBC_Excel"
```

2.7.5 MS Excel (2007)

To describe an MS Excel 2007 database (here the example database), type:

```
eDb2 <- edb( dbName = "soils.xlsx", dbType = "RODBC_Excel",
    excelVersion = 2007 )
```

`edb()` arguments are:

- `dbName`: Name and path of the database, as a text string;
- `dbType`: Database type. Here the string `RODBC_Excel` should be understood as "An Excel file managed (internally) by the RODBC package";
- `excelVersion`: Single integer. Excel version. Must be set to `2007` for MS Excel 2007. Can be ignored for earlier versions of MS Excel.

We now have an object called `eDb2` that can be used for all our database operations.

```
# Object class:
class( eDb2 )
```

```
[1] "edb"         "RODBC_Excel"
```

2.7.6 MySQL

It is a bit more complicated to describe a MySQL database. Type:

```
mDb <- edb(
    dbType       = "RODBC_MySQL",
    dbSourceName = "nameOfODBCSource", # or any name you like
    dbName       = "nameOfDatabase",
    dbLogin      = "yourUserName",
    dbPwd        = "yourPassword",
    dbHost       = "127.0.0.1",
    dbPort       = 3306
)   #
```

`edb()` arguments are:

- `dbSourceName`: Name of the ODBC connection to the database. That connexion does not need to exist yet. You can create it manually, or create it using `edbDataSource()` (see below);
- `dbName`: Name of the database in the MySQL system (not in ODBC);
- `dbType`: Database type. Here the string `RODBC_MySQL` should be understood as "A MySQL database managed (internally) by the RODBC package";
- `...`: Replace all the login, password and IP/Host values by values relevant for your database;

We now have an object called `mDb` that will be used for all our database operations. You need to register you database first in ODBC. You can do it with `edbDataSource()`, but please notice that this functions is experimental (be careful)[1]:

```
edbDataSource( mDb, verbose = TRUE )
```

We now have an object called `mDb` that can be used for all our database operations.

```
# Object class:
class( mDb )
```

```
[1] "edb"         "RODBC_MySQL"
```

2.8 Second step: 'inspect' you database (table and column names)

Notice: From this part of the tutorial, the examples will be given for the SQLite database only. Nonetheless, the R code would be exactly the same for other database type (MS Access, MS Excel, MySQL).

[1]The code below is not tested during vignette compilation. Just displayed

2.8.1 List tables

To list the tables in a database, type:

```
edbNames( edb = sDb )
[1] "HORIZON"         "MISCFORMAT"      "PROFILE"
[4] "WRB_SOIL_GROUP"  "sqlite_sequence"
```

You can get more details, set the argument `onlyNames = TRUE`, but that only works with MS Access, MS Excel and MySQL, not with SQLite.

2.8.2 Column names (in a table)

To list the columns in a table, use `edbColnames`[2]:

```
edbColnames( edb = sDb, tableName = "PROFILE" )
[1] "ID_PROFILE"        "NAME"              "STUDY"
[4] "ID_WRB_SOIL_GROUP" "LATITUDE"          "LONGITUDE"
[7] "COMMENTS"
```

2.8.3 Dimensions (of a table)

Similarly, you can obtain the number of row, the number of columns or the dimension of a table, use `edbNRow`, `edbNCol` or `edbDim`[3]:

```
# Number of rows:
edbNRow( edb = sDb, tableName = "PROFILE" )
[1] 2
# Number of columns:
edbNCol( edb = sDb, tableName = "PROFILE" )
[1] 7
# Dimenssions (rows x columns):
edbDim( edb = sDb, tableName = "PROFILE" )
[1] 2 7
```

2.9 Retreive and subset tables from a database

Now that we know which tables we have in the database, and which columns are in the table `"PROFILE"`, we can retrieve the data we need.

Retrieving data from a table in an `edb` database works a like data.frame's single square brackets subsetting, but not exactly like data.frame.

[2]Technical remark: It is not possible to use `colnames()`. This function is not generic, and can thus only be applied to data.frame and matrix. Moreover, `edb` class objects may contain several tables, thus the need for an additional argument concerning the name of the table

[3]Technical remark: It is not possible to use textttnrow(), `ncol()` or `dim()`. `edb` class objects may contain several tables, thus the need for an additional argument concerning the name of the table

For data.frame, the subsetting pattern is:

```
myDataFrame[ i, j ]
```

Where `i` and `j` are either indexes, names or a vector of logical indicating which rows and columns must be retreived, respectively.

For edb databases, we must also specify which table we are interested in, so the subsetting pattern is:

```
myEdbDatabase[ tableName, sRow, sCol ]
```

So we have 3 slots instead of 2, separated by commas. `tableName` is the name of the table (only one value), `sRow` is a list, containing one or several constrains to be applied for selecting some rows only (see below), and `sCol` is a vector of characters with the name of the columns to retrieve[4].

2.9.1 Retrieve a complete table

To get the table "PROFILE", type:

```
sDb[ "PROFILE" ]
  ID_PROFILE           NAME   STUDY ID_WRB_SOIL_GROUP LATITUDE
1          1 My 1st profile Study_A                19  59.8179
2          2 My 2nd profile Study_A                14  59.8150
  LONGITUDE    COMMENTS
1  17.66042 No comments
2  17.67393 No comments
```

If you don't like this subsetting style, you can use `edbRead()` instead:

```
edbRead( edb = sDb, tableName = "PROFILE" )
  ID_PROFILE           NAME   STUDY ID_WRB_SOIL_GROUP LATITUDE
1          1 My 1st profile Study_A                19  59.8179
2          2 My 2nd profile Study_A                14  59.8150
  LONGITUDE    COMMENTS
1  17.66042 No comments
2  17.67393 No comments
```

2.9.2 Retrieve a table subset (column constrains)

If you only want the columns "ID_PROFILE" and "NAME", type:

```
sDb[ "PROFILE", , c("ID_PROFILE","NAME") ]
  ID_PROFILE           NAME
1          1 My 1st profile
2          2 My 2nd profile
```

[4]Column subsetting using a vector of integers/indexes or logicals should be possible in the near future

If you don't like this subsetting style, you can use `edbRead()` instead:

```
edbRead(
    edb       = sDb,
    tableName = "PROFILE",
    sCol      = c("ID_PROFILE","NAME") )
```

```
  ID_PROFILE           NAME
1          1 My 1st profile
2          2 My 2nd profile
```

You can also use vector of indexes (integers) or vector of logicals to subset a table:

```
sDb[ "PROFILE", , c(1,2) ]
```

```
  ID_PROFILE           NAME
1          1 My 1st profile
2          2 My 2nd profile
```

```
#
sDb[ "PROFILE", , c(T,T,F,F,F,F,F) ]
```

```
  ID_PROFILE           NAME
1          1 My 1st profile
2          2 My 2nd profile
```

But indexes or logicals will not work for row subsetting (see below).

2.9.3 Retrieve a table subset (row constrains)

In the table "WRB_SOIL_GROUP", if you only want the row whose column "ID_WRB_SOIL_GROUP" are 1 to 10, type:

```
sDb[ "WRB_SOIL_GROUP", list( "ID_WRB_SOIL_GROUP" = 1:10 ) ]
```

```
   ID_WRB_SOIL_GROUP ABBREV        NAME
1                  1     AC     Acrisol
2                  2     AB Albeluvisol
3                  3     AL      Alisol
4                  4     AN     Andosol
5                  5     AT   Anthrosol
6                  6     AR    Arenosol
7                  7     CL    Calcisol
8                  8     CM    Cambisol
9                  9     CH   Chernozem
10                10     CR     Cryosol
```

As you can see, if we want to retrieve only some **rows**, we have to precise a constrain on some of the **columns**.

You can add as many column constrains as you want:

```
sDb[ "WRB_SOIL_GROUP",
    list( "ID_WRB_SOIL_GROUP" = 1:10,
          "ABBREV" = c("AL","AT","PL") ) ]
```

```
  ID_WRB_SOIL_GROUP ABBREV      NAME
1                 3     AL    Alisol
2                 5     AT Anthrosol
```

If you don't like this subsetting style, you can use `edbRead()` instead:

```
edbRead(
    edb       = sDb,
    tableName = "WRB_SOIL_GROUP",
    sRow      = list( "ID_WRB_SOIL_GROUP" = 1:10 ) )
```

```
   ID_WRB_SOIL_GROUP ABBREV        NAME
1                  1     AC     Acrisol
2                  2     AB Albeluvisol
3                  3     AL      Alisol
4                  4     AN     Andosol
5                  5     AT   Anthrosol
6                  6     AR    Arenosol
7                  7     CL    Calcisol
8                  8     CM    Cambisol
9                  9     CH   Chernozem
10                10     CR     Cryosol
```

```
#
# Multiple constrains (const1 AND const2):
edbRead(
    edb       = sDb,
    tableName = "WRB_SOIL_GROUP",
    sRow      = list( "ID_WRB_SOIL_GROUP" = 1:10,
                      "ABBREV" = c("AL","AT","PL") ) )
```

```
  ID_WRB_SOIL_GROUP ABBREV      NAME
1                 3     AL    Alisol
2                 5     AT Anthrosol
```

It is of course possible to use both row and column subsetting.

```
sDb[ "WRB_SOIL_GROUP", list( "ID_WRB_SOIL_GROUP" = 1:10 ),
    c("ID_WRB_SOIL_GROUP","NAME") ]
```

```
   ID_WRB_SOIL_GROUP        NAME
1                  1     Acrisol
2                  2 Albeluvisol
3                  3      Alisol
4                  4     Andosol
5                  5   Anthrosol
6                  6    Arenosol
7                  7    Calcisol
8                  8    Cambisol
9                  9   Chernozem
10                10     Cryosol
```

You can also add some SQL constrains in the sRow list, for instance:

```
sDb[ "WRB_SOIL_GROUP", list( "SQL" = "NAME LIKE 'Al%'" ) ]
```

```
  ID_WRB_SOIL_GROUP ABBREV        NAME
1                 2     AB Albeluvisol
2                 3     AL      Alisol
```

If you want to see how the arguments are internally translated into SQL queries use the argument `verbose = TRUE`.

```
sDb[ "WRB_SOIL_GROUP", list( "ID_WRB_SOIL_GROUP" = 1:10 ),
    c("ID_WRB_SOIL_GROUP","NAME"), verbose = TRUE ]
```

```
SQL statement:
SELECT ID_WRB_SOIL_GROUP, NAME
FROM [WRB_SOIL_GROUP]
WHERE ([ID_WRB_SOIL_GROUP] IN (1, 2, 3, 4, 5, 6, 7, 8, 9, 10));
```

```
   ID_WRB_SOIL_GROUP        NAME
1                  1     Acrisol
2                  2 Albeluvisol
3                  3      Alisol
4                  4     Andosol
5                  5   Anthrosol
6                  6    Arenosol
7                  7    Calcisol
8                  8    Cambisol
9                  9   Chernozem
10                10     Cryosol
```

Finally, you can chose to combine row constrains with **OR** instead of **AND** (the default), by using the argument `sRowOp = "OR"`. Complex combinations of AND and OR are not possible unfortunately (you have to use RSQLite or RODBC queries).

```
sDb[ "WRB_SOIL_GROUP", list( "ID_WRB_SOIL_GROUP" = 1:10,
    "ABBREV" = c("AL","AT","PL") ), sRowOp = "OR" ]
```

```
   ID_WRB_SOIL_GROUP ABBREV        NAME
1                  1     AC     Acrisol
2                  2     AB Albeluvisol
3                  3     AL      Alisol
4                  4     AN     Andosol
5                  5     AT   Anthrosol
6                  6     AR    Arenosol
7                  7     CL    Calcisol
8                  8     CM    Cambisol
9                  9     CH   Chernozem
10                10     CR     Cryosol
11                23     PL    Planosol
```

2.10 Write data

First lets get some data we want to write in the database:

```
profileTbl <- sDb[ "PROFILE" ]
profileTbl
```

```
  ID_PROFILE           NAME   STUDY ID_WRB_SOIL_GROUP LATITUDE
1          1 My 1st profile Study_A                19  59.8179
2          2 My 2nd profile Study_A                14  59.8150
  LONGITUDE    COMMENTS
1  17.66042 No comments
2  17.67393 No comments
```

2.10.1 Append data to a table:

We will try to write back / add this table to the table "PROFILE", as if it was new data. But we need first to change the field "ID_PROFILE", as it is a PRIMARY KEY in the database (no duplicates allowed):

```
profileTbl[, "ID_PROFILE" ] <- 3:4
```

Now the table can be written:

```
sDb[ "PROFILE" ] <- profileTbl
```

This is the same as `sDb[ "PROFILE", mode = "a" ] <- profileTbl`, which mean we have *a*ppend the data at the end of the table.

Another way to do the same operation is:

```
# First change the IDs:
profileTbl[, "ID_PROFILE" ] <- 5:6
#
edbWrite( edb = sDb, tableName = "PROFILE", data = profileTbl )
```

```
[1] TRUE
```

We can check that the data has been written:

```
sDb[ "PROFILE", , c("ID_PROFILE","NAME") ]
```

```
  ID_PROFILE           NAME
1          1 My 1st profile
2          2 My 2nd profile
3          3 My 1st profile
4          4 My 2nd profile
5          5 My 1st profile
6          6 My 2nd profile
```

2.10.2 Update data in a table:

If we want to update data in a table, we need to use `edbWrite()` (the square bracket method will not work). We have to set the argument `mode = "u"` and also the argument `pKey = "ID_PROFILE"` to state that pKey must be used as a key to identify the fields that must be updated (it does not have to be a PRIMARY key in the database, but it is better if it is a unique indentifier).

But first lets modify the data we will write:

```
profileTbl[,"NAME"] <- c("My 5th profile","My 6th profile")
```

Then we can update this in the database:

```
edbWrite(
    edb       = sDb,
    tableName = "PROFILE",
    data      = profileTbl,
    mode      = "u",
    pKey      = "ID_PROFILE" )
NULL
```

And check what has been written:

```
sDb[ "PROFILE", list( "ID_PROFILE" = 5:6 ),
    c("ID_PROFILE","NAME") ]
  ID_PROFILE           NAME
1          5 My 5th profile
2          6 My 6th profile
```

2.11 Delete records and drop tables

2.11.1 Delete records in a table

The function `edbDelete()` allows to delete all or a selection of records in a given table. See `?edbWrite` help page for an example and `?edbDelete` for the list of possible arguments for this function.

The function `edbDelete()` is not supported by (RODBC) MS Excel.

WORK ON PROGRESS!

2.11.2 Drop a table in a database

The function `edbDrop()` allows to drop / delete tables in a database. See `?edbWrite` help page for an example and `?edbDrop` for the list of possible arguments for this function.

The function `edbDrop()` is partially supported by (RODBC) MS Excel: All rows are deleted, but not the table!.

WORK ON PROGRESS!

3 Advanced usage

3.1 Fetch AUTOINCREMENT primary keys attributed by the database

When working with relational databases in which referential integrity is important, tables are attributed a primary key, that has to be unique for each record. You may decide yourself what is the value of this primary key, but it is sometimes impossible (when several processes or computers write at the same time on the table for instance). For this reason the primary is often set as `AUTOINCREMENT` (or `AUTO_INCREMENT`), meaning that the database can attribute automatically a primary key if it has not been provided by the user. When using that feature, it is important be be able to know what primary key has been attributed to the record we just wrote in the database (again, several processes may have written at the same time on the database). SQL databases have special functions for this (like `LAST_INSERT_ID()` in MySQL). These functions have been implemented in `edbWrite()` to make it possible to retrieve the last inserted ID in R, without typing complicated SQL code (it does it for you).

This function is not supported by MS Excel (as Excel does not have primary keys).

The example below shows how to fetch the last inserted IDs. You need to specify the primary key using `getKey`:

WORK ON PROGRESS!

3.2 'On-the-fly' transformation of variables when reading from or writing to the database

See the argument `formatCol` in `edbWrite()`...

WORK ON PROGRESS!

3.3 About dates and boolean in SQLite databases

See the argument `formatCol` in `edbWrite.RSQLite_SQLite()`...

WORK ON PROGRESS!

3.4 Database operation log table

See the argument `logOp` and `logMsg` in `edbWrite()`, `edbDelete()` and `edbDrop()`...

WORK ON PROGRESS!

```
[1] TRUE
```

4 Misc: Session info

Information on R Session and packages versions (that were used to build this vignette):

```
sessionInfo()
```

```
R version 3.1.2 Patched (2014-11-14 r66984)
Platform: x86_64-unknown-linux-gnu (64-bit)

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8        LC_COLLATE=C
 [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods
[7] base

other attached packages:
[1] easyrsqlite_0.7.4 RSQLite_1.0.0     DBI_0.3.1
[4] easydb_0.7.5

loaded via a namespace (and not attached):
[1] tools_3.1.2
```

```
packageVersion( pkg = "easydb" )
```

```
[1] âĂŸ0.7.5âĂŹ
```

```
# packageVersion( pkg = "RODBC" )
packageVersion( pkg = "DBI" )
```

```
[1] âĂŸ0.3.1âĂŹ
```

```
packageVersion( pkg = "RSQLite" )
```

```
[1] âĂŸ1.0.0âĂŹ
```

References

[1] David˜A. James. *RSQLite: SQLite interface for R*, 2010. R package version 0.9-2.

[2] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2011. ISBN 3-900051-07-0.

[3] Brian Ripley, , and from 1999˜to Oct 2002 Michael˜Lapsley. *RODBC: ODBC Database Access*, 2010. R package version 1.3-2.