Package: kwb.db (via r-universe)

August 21, 2024

Title Functions supporting data base access

Version 0.7.0

Description This package contains some useful functions, especially for simplifying data transfer between MS Access databases and R. With the functions of this package it is not needed any more to open and close a database connection explicitely; this is done 'behind the scenes' in the functions. Instead of a database connection the path to the database file needs to be passed to the functions as an argument. The main functions are hsGetTable and hsPutTable which transfer data from an MS Access database to a data frame in R and save data from a data frame in R into a table in an MS Access database, respectively. Take care when getting time series data from an MS Access database, see therefore hsMdbTimeSeries. Use hsTables to get a list of tables that are available in a database and hsFields to get a list of table fields that are contained in a database table.

License MIT + file LICENSE

URL https://github.com/KWB-R/kwb.db

BugReports https://github.com/KWB-R/kwb.db/issues

Imports kwb.datetime (>= 0.4.0), kwb.utils (>= 0.4.4), odbc32 (>= 0.2.7), RODBC (>= 1.3.16)

Suggests testthat (>= 2.2.1), xml2

Remotes github::cran/RODBC@1.3-16, github::hsonne/odbc32@r3.0.0, github::kwb-r/kwb.datetime, github::kwb-r/kwb.utils

Encoding UTF-8

RoxygenNote 7.2.3

Repository https://kwb-r.r-universe.dev

RemoteUrl https://github.com/KWB-R/kwb.db

RemoteRef HEAD

RemoteSha d4b0d9b2a91a07e0dc5e7554dbf7f5c06ca51486

Contents

`connectionStringAccess` *Connection String Access*

Description

Connection String Access

Usage

```
connectionStringAccess(mdb, uid = "", pwd = "", globalPartialBulkOps = 0)
```

Arguments

mdb	full path to MS Access file
uid	user id, if any
pwd	password, if any
globalPartialBulkOps	A Long value (read/write) that determines the behavior of the Jet database engine when SQL DML bulk operations fail. When set to allow partial completion of bulk operations, inconsistent changes can occur, because operations on some records could succeed and others could fail. When set to allow no partial completion of bulk operations, all changes are rolled back if a single error occurs. The Jet OLEDB:Global Partial Bulk Ops property setting can be overridden for the current Recordset object by setting the Jet OLEDB:Partial Bulk Ops property.The Jet OLEDB:Global Partial Bulk Ops and Jet OLEDB:Partial Bulk Ops properties can be set to any of the following values: Default = 0, Partial = 1, No Partial = 2

References

http://msdn.microsoft.com/en-us/library/office/aa140022%28v=office.10%29.aspx

---

createRDatabaseInterface

Create R-Function Interface to Database

Description

Create R-Function Interface to Database

Usage

```
createRDatabaseInterface(
  db = NULL,
  dbSchema = NULL,
  tableNamePattern = "",
  functionPrefix = "db_",
  pattern = "^",
  rfile = file.path(tempdir(), "dbInterface.R"),
  rfile.create = file.path(tempdir(), "dbInterfaceCreate.R"),
  create.create.functions = FALSE,
  dbg = TRUE,
  sql.dbg = TRUE
)
```

Arguments

db	name of ODBC data source or full path to MS Access or Excel file
dbSchema	list structure describing the database schema, as returned by kwb.odm::dbSchema_ODM()
tableNamePattern	pattern matching the names of tables/views for which an accessor function is to be generated
functionPrefix	prefix to be given to each generated function. Default: "db_"
pattern	pattern matching the part of the table name that will be replaced with *functionPrefix*. Default: "^", i.e. the prefix will be appended to the start of the table name
rfile	full path to the file in which to store the data access functions
rfile.create	full path to the file in which to store the data creation functions (only relevant if *crete.create.functions* is TRUE)
create.create.functions	logical. If TRUE (the default if FALSE) not only data access but also data creation functions are generated
dbg	if TRUE, progress messages are shown during the creation of the interface functions
sql.dbg	if TRUE, SQL strings used to access the database are shown when calling the interface functions

---

currentDb *Get Path to or Name of Current Database*

Description

Get Path to or name of current database (as set with setCurrentDb)

Usage

```
currentDb(dbg = TRUE)
```

Arguments

dbg	if TRUE, a message obout setting the current database is printed

dataFrameToSqlTuples *Create SQL Tuples from Data Frame*

Description

Create SQL Tuples from Data Frame

Usage

```
dataFrameToSqlTuples(newData)
```

Arguments

newData	a data frame

Value

vector of character strings each of which represents one row in newData

Examples

```
x <- data.frame(
  name = c("Peter", "Paul"),
  birthday = as.POSIXct(c("1981-12-13", "2003-01-16"))
)

setCurrentSqlDialect("msaccess")
dataFrameToSqlTuples(x)

# Note that the representation of a date and time is different in MySQL
setCurrentSqlDialect("mysql")
dataFrameToSqlTuples(x)
```

dumpDatabase *Export Database Tables to CSV Files*

Description

Exports all tables of a database of which the names match a given pattern to csv files.

Usage

```
dumpDatabase(
  db,
  pattern = "^tbl",
  target_dir = NULL,
  create_target_dir = FALSE,
  sep = ",",
  dec = ".",
  as.is = FALSE,
  qmethod = "double",
  row.names = FALSE,
  ...
)
```

Arguments

db	full path to database or name of ODBC data source
pattern	pattern matching names of tables to be exported. Default: "^tbl", i.e. tables starting with "tbl"
target_dir	target directory. By default a new directory is created in the same directory as mdb resides in. The new directory has the same name as the database file with dots substituted with underscores
create_target_dir	if TRUE, the target directory tdir is created if it does not exist.
sep	passed to write.table
dec	passed to write.table
as.is	passed to sqlGetResults. If TRUE (the default is FALSE), original data types are kept when the table is read into R. By default the types are converted to appropriate R data types (e.g. dates are converted from strings to date objects).
qmethod	passed to write.table
row.names	passed to write.table
...	further arguments passed to write.table

---

getCurrentSqlDialect *Get Current SQL Dialect*

Description

Get Current SQL Dialect

Usage

```
getCurrentSqlDialect(warn = TRUE, dbg = FALSE)
```

Arguments

warn	if TRUE and if no current SQL dialog is stored in the options, the program stops with an error message
dbg	if TRUE, a message about the current SQL dialect is printed

---

getDatabaseFieldInfo *Get Information on Table Field Names and Types*

Description

Get Information on Table Field Names and Types

Usage

```
getDatabaseFieldInfo(db)
```

Arguments

db	path to MS Access or MS Excel file or name of ODBC data source

Value

data frame with columns TABLE_NAME, COLUMN_NAME, TYPE_NAME, DECIMAL_DIGITS

---

getDatabaseSchema *Get Database Schema*

Description

Get Database Schema

Usage

```
getDatabaseSchema(db, tableNames = NULL, tableTypes = c("TABLE", "VIEW"))
```

Arguments

db	full path to database (*.mdb, *.xls) or name of ODBC database
tableNames	optional. Vector of table names of tables to be included
tableTypes	types of database objects to be included. Default: c("TABLE", "VIEW")

Value

list with elements *tables* and *relationships*. Element *tables* is a list o named elements with the name representing the table names and the elements being lists describing the table...

getFilteredRecords *Get Filtered Records*

Description

Get Filtered Records

Usage

```
getFilteredRecords(db, tableName, keyValues, fields, like, ...)
```

Arguments

db	database name or database file
tableName	name of the table from which to read records
keyValues	list of key = value pairs with the keys representing field names of the table tableName, defining filter criteria
fields	table fields to be selected
like	if TRUE the SQL LIKE operator is used instead of the equal operator = when matching the values in keyValues with the table fields
...	additonal arguments passed to selectFromDb

getNamedExcelRanges *Get Named Ranges from Excel File*

Description

Get Named Ranges from Excel File

Usage

```
getNamedExcelRanges(
  file,
  pattern = "^range",
  dbg = TRUE,
  stringsAsFactors = FALSE,
  ...
)
```

Arguments

file	path to Excel file
pattern	pattern matching the names of named cell ranges to be read. By default, all ranges starting with range are read.
dbg	logical. If TRUE debug messages are shown
stringsAsFactors	passed to hsGetTable
...	further arguments passed to hsGetTable

Value

list of data frames each of which represents the content a named cell range in the Excel file and each of which was read with hsGetTable.

---

getSqlDialect *Get SQL Dialect from Given Database*

Description

get SQL dialect ("mysql" or "msaccess") from given database

Usage

```
getSqlDialect(db, use2007Driver = NULL)
```

Arguments

db	ODBC database name or full path to database (mdb or xls)
use2007Driver	passed to isMySQL

---

getTypeIdentifier *Get Type Identifier*

Description

Get Type Identifier

Usage

```
getTypeIdentifier(x)
```

Arguments

x	R object for which to find an adequate database object type

Value

Returns "int", "double", "text", "date_time" or "boolean" depending on the data type of *x*

`hsClearTable` *Clear a Database Table*

Description

Deletes all the rows of the table *tbl*. This function performs opening of the connection, clearing of the table and closing of the connection. If an error occurs the program stops and an error description is shown.

Usage

```
hsClearTable(mdb, tbl, cond = TRUE, ...)
```

Arguments

`mdb`	full path to MS Access database (*.mdb).
`tbl`	table name.
`cond`	optional: condition.
`...`	additional arguments passed to hsSqlQuery, e.g. "errors=TRUE"

See Also

`hsDropTable`

`hsCloseDb` *Close Connection to MS Access or Excel*

Description

Closes the database connection.

Usage

```
hsCloseDb(con)
```

Arguments

`con`	Open database connection as returned by hsOpenDb/`odbcConnect`

See Also

`hsOpenDb`

hsCloseMdb *Close Connection*

Description

Deprecated. Use codehsCloseDb instead.

Usage

```
hsCloseMdb(con)
```

Arguments

con	Open database connection as returned by hsOpenMdb/odbcConnect

hsDropTable *Drop Database Table(s)*

Description

Removes the table *tbl* (if permitted). This function performs opening of the connection, dropping of the table and closing of the connection. If an error occurs the program stops and an error description is shown.

Usage

```
hsDropTable(mdb, tbl, isPtrn = FALSE, dbg = TRUE)
```

Arguments

mdb	full path to MS Access database (*.mdb).
tbl	table name.
isPtrn	if TRUE, *tbl* is interpreted as a regular expression matching the names of the tables to be deleted.
dbg	if TRUE, debug messages are shown

See Also

hsClearTable

hsDumpMdb *Export Database Tables to CSV Files*

Description

Deprecated. Please use dumpDatabase instead.

Usage

```
hsDumpMdb(
  mdb,
  ptrn = "^tbl",
  tdir = file.path(dirname(mdb), gsub("\\.", "_", basename(mdb))),
  create_target_dir = FALSE
)
```

Arguments

mdb	full path to database
ptrn	pattern matching names of tables to be exported. Default: "^tbl", i.e. tables starting with "tbl"
tdir	target directory. By default a new directory is created in the same directory as mdb resides in. The new directory has the same name as the database file with dots substituted with underscores
create_target_dir	if TRUE, the target directory tdir is created if it does not exist.

hsFields *Available Fields in Database Table*

Description

Returns a vector containing the field names of a database table.

Usage

```
hsFields(
  mdb,
  tbl,
  namesOnly = TRUE,
  chopDollar = TRUE,
  ignore.case = (!isMySQL(mdb, use2007Driver = use2007Driver)),
  use2007Driver = NULL,
  dbg = FALSE
)
```

Arguments

mdb	full path to MS Access database file (extension ".mdb" or ".accdb") or MS Excel file (extension ".xls" or ".xlsx").
tbl	table name.
namesOnly	if TRUE, only field names are returned, otherwise all available information on the fields. Default: TRUE
chopDollar	if TRUE (default), a dollar sign at the end of the table name is removed before sending it to sqlColumns,
ignore.case	if TRUE, case is ignored when comparing the given table with the names of the existing tables. Default: FALSE
use2007Driver	passed to isMySQL
dbg	if TRUE, debug messages are shown

Value

Vector containing the field names of the database table (if namesOnly = TRUE) or data.frame with columns *COLUMN_NAME*, *DATA_TYPE*, *TYPE_NAME*, *COLUMN_SIZE*, *BUFFER_LENGTH*, *DECIMAL_DIGITS*, *NUM_PREC_RADIX*, *NULLABLE* describing the database fields in detail, otherwise.

See Also

hsTables

Examples

```
## Not run:
## List the fields of table "tbl_Hyd" in the example database
## (only on Windows!)

if (.Platform$OS.type == "windows") {
  setCurrentSqlDialect("msaccess")
  fields <- hsFields(xmdb(), "tbl_Hyd")
  fields
}

## Ouput:
# [1] "Zeitst"  "Q"       "v"       "H"       "T_Kanal"

## End(Not run)
```

hsGetTable *Get Table from MS Access Database*

Description

Provides data from an MS Access database table in forms of a data frame.

Usage

```
hsGetTable(
  mdb,
  tbl,
  cond = "TRUE",
  fields = "*",
  dbg = TRUE,
  check = TRUE,
  use2007Driver = NULL,
  ...
)
```

Arguments

mdb	full path to MS Access database file (extension ".mdb" or ".accdb") or MS Excel file (extension ".xls" or ".xlsx").
tbl	Table name. Put it into brackets [] if it contains spaces and if it mdb does not point to a MySQL database
cond	Condition string.
fields	Comma separated list of names of fields to be selected.
dbg	if TRUE, debug messages are shown, else not
check	if TRUE (default), *tbl* is checked for existence in *mdb* before trying to get the data and a list of available tables is shown in the case that the table does not exist.
use2007Driver	passed to hsTables and hsSqlQuery
...	Additional arguments to be passed to hsSqlQuery

Details

ATTENTION: This function may not return what you want if the table contains a timestamp field. Use hsMdbTimeSeries instead.

Value

data.frame containing data from table in database

See Also

hsSqlQuery, hsPutTable, hsGetTimeSeries, hsMdbTimeSeries

Examples

```
## Not run:
## Get all datasets from tbl_Hyd in example database where
## Q > 1.0 m3/s and temperature > 20 degree Celsius
## (only on Windows!)

if (.Platform$OS.type == "windows") {
  ts <- hsGetTable(xmdb(), "tbl_Hyd", "Q > 1.0 AND T_Kanal > 20")
  head(ts)
}

## Output:
# Zeitst     Q     v     H T_Kanal
# 1 2011-08-24 22:33:00 1.075 0.459 1.366    20.1
# 2 2011-08-24 22:34:00 1.062 0.453 1.370    20.2
# 3 2011-08-24 22:35:00 1.050 0.449 1.364    20.2
# 4 2011-08-24 22:36:00 1.042 0.446 1.361    20.3
# 5 2011-08-24 22:37:00 1.032 0.443 1.354    20.3
# 6 2011-08-24 22:38:00 1.010 0.436 1.348    20.4

## TAKE CARE when getting time-series data:
if (.Platform$OS.type == "windows") {
  hsGetTable(xmdb(), "tblTimestampTest_DST")
}

## Output:
#                tstamp
# 1 2011-03-27 01:00:00
# 2 2011-03-27 01:30:00
# 3                <NA>
# 4                <NA>
# 5 2011-03-27 03:00:00
# 6 2011-03-27 03:30:00

## As the output shows the timestamps between 02:00:00 and
## 02:59:59 have been set to <NA>. Reason: When retrieving
## date/time data from MS Access, R converts the timestamps
## from a text representation into POSIXct objects. As POSIXct's
## standard time zone seems to be taken from the Windows system
## R tries to convert to Central European Time (CET) which
## does not exist for the hour in which time is switched to
## daylight-saving time (as in the example).

## This standard behaviour can be changed by setting the
## standard time zone:
tz <- Sys.getenv("tz") # save current standard time zone
Sys.setenv(tz = "UTC") # set standard time zone to UTC
```

```
## The same command as above now delivers all timestamps
## (in Coordinated Universal Time, UTC):

if (.Platform$OS.type == "windows") {
  hsGetTable(xmdb(), "tblTimestampTest_DST")
}

## Output:
#                tstamp
# 1 2011-03-27 01:00:00
# 2 2011-03-27 01:30:00
# 3 2011-03-27 02:00:00
# 4 2011-03-27 02:30:00
# 5 2011-03-27 03:00:00
# 6 2011-03-27 03:30:00

## Reset standard time zone
Sys.setenv(tz = tz)

## End(Not run)
```

`hsGetTimeSeries` *Get Time Series With Timestamp Info*

Description

Reads time-series data from an MS Access database table and returns a data frame containing the data. In the data frame the timestamp column contains the timestamps as they are converted to by R from (text versions of) the original timestamps read from MS ACCESS. As this conversion may fail (e.g. the time information gets lost when transferring timestamps from large data sets between R and MS Access) this function may return different pieces of information on the timestamp in forms of additional columns, preceding the timestamp column, in the result data frame. Per default, eleven additional columns are returned: 1. <ts>_txt (timestamp as text), 2. <ts>_Date (date only), 3. <ts>_dSince18991230 (number of days since 1899-12-30), 4. <ts>_secInDay (number of seconds within the day), 5. <ts>_minInDay (number of minutes within the day), 6. <ts>_year (year), 7. <ts>_month (number of month), 8. <ts>_day (number of day within the month), 9. <ts>_h (hours within day), 10. <ts>_min (minutes within hour), 11. <ts>_s (seconds within minute) where in each case <ts> is the name of the timestamp field.

Usage

```
hsGetTimeSeries(
  mdb,
  tbl,
  tsField = hsTsField(mdb, tbl),
  fields = "*",
  minDate = NULL,
  maxDate = NULL,
```

```
  xTsFields = c(1:11),
  inclLast = TRUE,
  sqlFilter = "TRUE",
  dbg = FALSE
)
```

Arguments

mdb	Full path to MS Access database file (*.mdb)
tbl	Name of table containing the time-series data.
tsField	Name of table field containing the timestamps.
fields	Vector containing names of value fields to be selected from the table. This vector may or may not contain the name of the timetamp field.
minDate	Minimum date (and time) of time interval to be selected in ISO-Syntax: yyyy-mm-dd [HH:MM:SS], where the part in brackets in optional.
maxDate	Day following the maximum date of the time interval to be selected, in ISO-Syntax: yyyy-mm-dd [HH:MM:SS], where the part in brackets in optional.
xTsFields	Extra timestamp fields to be selected. Vector containing numbers between 1 and 11, where each number represents a type of date/time information as described for function hsSqlExTsFields.
inclLast	If TRUE, *maxDate* will be included in result data set, else excluded.
sqlFilter	additional SQL filter criterion
dbg	if TRUE, debug messages are shown

Details

This function is called internally by the higher-level function hsMdbTimeSeries that reconstructs the correct timestamps from the different pieces of timestamp information and provides them in forms of POSIXct objects in UTC timezone.

Use hsMdbTimeSeries instead if you do not want to care about any timestamp conversion problems!

Value

data frame containing the requested data (timestamp and value columns) and additional columns preceding the timestamp column containing different pieces of information on the timestamp.

See Also

hsMdbTimeSeries, hsGetTable, hsSqlExTsFields

Examples

```
## Not run:
## Get flow time series of 24 of July 2011 from tbl_Hyd in example database
## Additionally to the timestamp that is created by R, return the date only
## (timestamp info id = 2) and the number of minutes within the day
## (timestamp info id = 5).
```

```
setCurrentSqlDialect("msaccess")

if (.Platform$OS.type == "windows") {
  ts <- hsGetTimeSeries(
    mdb = xmdb(),
    tbl = "tbl_Hyd",
    tsField = "Zeitst",
    fields = c("Q", "v"),
    minDate = "2011-08-24",
    maxDate = "2011-08-25",
    xTsFields = c(2, 5),
    dbg = TRUE
  )

  ## Show the last records of the returned dataset.
  tail(ts)
}

## Output:
#      Zeitst_Date Zeitst_minInDay              Zeitst     Q     v
# 1435  2011-08-24            1435 2011-08-24 23:55:00 0.638 0.281
# 1436  2011-08-24            1436 2011-08-24 23:56:00 0.601 0.265
# 1437  2011-08-24            1437 2011-08-24 23:57:00 0.564 0.249
# 1438  2011-08-24            1438 2011-08-24 23:58:00 0.536 0.237
# 1439  2011-08-24            1439 2011-08-24 23:59:00 0.504 0.223
# 1440  2011-08-25               0 2011-08-25 00:00:00 0.483 0.214

## End(Not run)
```

`hsJetDate` *Date to "mm/dd/yyyy HH:MM:SS"*

Description

Returns a date in MS Jet SQL syntax: mm/dd/yyyy HH:MM:SS

Usage

```
hsJetDate(datetime, dbg = FALSE)
```

Arguments

`datetime`	Date (and time) information in forms of Date object or POSIX object or string.
`dbg`	if TRUE, debug messages are shown

hsLookupOrAddRecord *Lookup or Add Record*

Description

Looks up a record in a database table and returns the ID of the record. If the record is not found it is inserted to the table and the ID of the new record is returned.

Usage

```
hsLookupOrAddRecord(
  mdb,
  tbl,
  keyAssigns,
  fieldAssigns = NULL,
  idField = hsFields(mdb, tbl)[1],
  dbg = FALSE
)
```

Arguments

mdb	full path to MS Access database
tbl	name of table in which record is to be looked up
keyAssigns	key-value-assignements used to identify the record to be looked up. The assignments are defined in forms of a list, e.g. list(key1 = "value1", key2 = "value2").
fieldAssigns	further field-value-assignements used when a new record needs to be inserted. The assignments are defined in forms of a list, e.g. list(field1 = "value1", field2 = "value2").
idField	name of ID field, default: name of first table field
dbg	if TRUE, debug messages are shown

hsMdbTimeSeries *Get Mdb time series in UTC*

Description

Reads time-series data from an MS Access database table and returns a data frame containing the data. The name of the timestamp field must be given in *tsField* and the names of the value fields to be selected from the table must be given in vector *fields*. Instead of an ODBC channel the name of the database must be given. This function takes care that the timestamps are transferred correctly between MS Access and R by requesting date and time information separately from MS Access and putting both together to a POSIXct object in UTC timezone. This is necessary because with very long data sets the RODBC function sqlQuery (or the function hsSqlQuery that calls this function) may deliver timestamps in which time information is lacking!

Usage

```
hsMdbTimeSeries(
  mdb,
  tbl,
  tsField = hsTsField(mdb, tbl),
  fields = "*",
  minDate = NULL,
  maxDate = NULL,
  resolution = "min",
  inclLast = TRUE,
  sqlFilter = "TRUE",
  dbg = FALSE,
  calcType = 1
)
```

Arguments

mdb	Full path to MS Access database file (*.mdb)
tbl	Name of table containing the time-series data.
tsField	Name of table field containing the timestamps.
fields	Vector containing names of value fields to be selected from the table. This vector may or may not contain the name of the timetamp field.
minDate	Minimum date (and time) of time interval to be selected in ISO-Syntax: yyyy-mm-dd [HH:MM:SS], where the part in brackets in optional.
maxDate	Day following the maximum date of the time interval to be selected, in ISO-Syntax: yyyy-mm-dd [HH:MM:SS], where the part in brackets in optional.
resolution	time resolution: "min" = minutes, "s" = seconds. If time resolution is "min" timestamps are rounded to the next full minute.
inclLast	If TRUE, *maxDate* will be included in result data set, else excluded.
sqlFilter	additional SQL filter criterion
dbg	if TRUE, debug messages are shown
calcType	for internal use only, do not change!

Value

data.frame with POSIXct timestamp column <strTimestamp> (UTC time zone) and value columns as selected in <strFieldList>

See Also

hsGetTimeSeries, hsGetTable

Examples

```
## Not run:
## Get flow time series of 24 of August 2011 from tbl_Hyd in example database

if (.Platform$OS.type == "windows") {

  ts <- hsMdbTimeSeries(
    xmdb(), "tbl_Hyd", "Zeitst", c("Q", "v"), "2011-08-24", "2011-08-25"
  )

  ## Show the last records of the returned dataset.
  tail(ts)
}

## Output:
#                   Zeitst     Q     v
# 1435 2011-08-24 23:55:00 0.638 0.281
# 1436 2011-08-24 23:56:00 0.601 0.265
# 1437 2011-08-24 23:57:00 0.564 0.249
# 1438 2011-08-24 23:58:00 0.536 0.237
# 1439 2011-08-24 23:59:00 0.504 0.223
# 1440 2011-08-25 00:00:00 0.483 0.214

## End(Not run)
```

---

hsOpenDb *Open Connection to MS Access or Excel*

Description

Opens database connection to MS Access or MS Excel, checks the connection and stops on error.

Usage

```
hsOpenDb(src, use2007Driver = NULL, dbg = FALSE, DBMSencoding = "", ...)
```

Arguments

src	full path to MS Access database file (extension ".mdb" or ".accdb") or MS Excel file (extension ".xls" or ".xlsx") or name of ODBC data source.
use2007Driver	if TRUE the functions odbcConnectAccess2007 and odbcConnectExcel2007 are used instead of odbcConnectAccess and odbcConnectExcel, respectively
dbg	if TRUE and if the connection could be established details of the connection are shown.
DBMSencoding	finally passed to odbcDriverConnect. Default: "", You may want to use: "UTF-8"
...	additional arguments passed to the odbcConnect...()-function

Value

On success an object of class RODBC describing the connection is returned. On failure nothing is returned and program execution stops.

See Also

hsCloseDb

Examples

```
## Not run:
## Open a connection to the example database
## (only on Windows!)

if (.Platform$OS.type == "windows") {

  con <- hsOpenDb(xmdb())
  con
}

## Details on the established connection are shown:
# RODBC Connection 9
# Details:
#   case=nochange
#   DBQ=C:\Users\hsonne\Documents\R\win-library\2.14\kwb.base\...
#   Driver={Microsoft Access Driver (*.mdb)}
#   DriverId=25
#   FIL=MS Access
#   MaxBufferSize=2048
#   PageTimeout=5
#   UID=admin

## Close the connection again

if (.Platform$OS.type == "windows") {

  hsCloseDb(con)
}

## End(Not run)
```

---

hsOpenMdb *Open Connection to MS Access Database*

Description

Deprecated. Use hsOpenDb instead.

Usage

```
hsOpenMdb(mdb, dbg = FALSE)
```

Arguments

mdb	full path to MS Access database file.
dbg	if TRUE and if the connection could be established details of the connection are shown.

---

hsPutTable *Save Data Frame to Database Table*

Description

Writes data to a database table. This function performs opening of the connection, saving of the data to a table and closing of the connection. If an error occurs the program stops and an error description is shown. If a table named *tbl* already exists in the database *mdb* the existing table is only overwritten if *overwrite* is TRUE. Otherwise a `hsSafeName` will be found for the table.

Usage

```
hsPutTable(
  mdb,
  myData,
  tbl = "tblTmp",
  types = NULL,
  overwrite = FALSE,
  DBMSencoding = "",
  dbg = TRUE
)
```

Arguments

mdb	full path to MS Access database file (*.mdb)
myData	data.frame containing data to be written to database table
tbl	Name of table to be created in the database
types	field types to be passed to sqlSave as argument *varTypes*, see ?sqlSave for details.
overwrite	shall existing table be overwritten?
DBMSencoding	finally passed to `odbcDriverConnect`. Default: "", You may want to use: "UTF-8"
dbg	if TRUE, debug messages are shown

Value

In case of success the name of the created table is returned.

See Also

hsSqlQuery, hsGetTable

Examples

```
## Not run:
## Create a data.frame
df1 <- data.frame(id = 1:4, rnd = sample(1:100)[1:4])

## Write data.frame into a table in the example database; as no
## table name is specified, a table name is generated. The name
## of the table is returned.
## (only on Windows!)

if (.Platform$OS.type == "windows") {

  tbl <- hsPutTable(xmdb(), df1)
  tbl # table name here: [1] "tblTmp"


  ## Get the data from the created table back again and print the
  ## data. As we see, a table containing four different random
  ## numbers between one and 100 has been created.
  df2 <- hsGetTable(xmdb(), tbl)
  df2
}

## Output:
#   id rnd
# 1  1  82
# 2  2  14
# 3  3  99
# 4  4   6

## End(Not run)
```

---

hsSetForeignKey *Set Foreign Key for Table Field*

Description

Set foreign key constraint for a table field

Usage

```
hsSetForeignKey(
  mdb,
  tbl,
```

```
  field,
  ref.tbl,
  ref.field,
  key.name = .getForeignKeyName(tbl, field, ref.tbl, ref.field),
  dbg = FALSE
)
```

Arguments

mdb	full path to MS Access database file (*.mdb)
tbl	name of table containing the field for which the foreign key constraint is to be defined
field	name of table field for which the foreign key constraint is to be defined
ref.tbl	name of table containing the referenced foreign key field
ref.field	name of foreign key field in ref.tbl
key.name	optional. Name to be given to the foreign key
dbg	passed to hsSqlQuery

---

hsSetPrimaryKey *Set Primary Key of Database Table*

Description

Sets fields with names given in vector *keyFields* as key fields of table *tbl* in MS ACCESS database *mdb*.

Usage

```
hsSetPrimaryKey(mdb, tbl, keyFields, dbg = FALSE)
```

Arguments

mdb	Full path to MS Access database file (*.mdb).
tbl	Name of table in which key fields are to be defined.
keyFields	(Vector of) key field name(s)
dbg	if TRUE, debug messages are shown

hsSqlExAnd *SQL Expression "AND"*

Description

Creates a boolean expression of the form
`bFunc(x1) AND bFunc(x2) AND ... AND bFunc(xn)`. This function can be used to create SQL queries where many table fields have to be checked in the same way for some criterion (see example).

Usage

```
hsSqlExAnd(x, bFunc)
```

Arguments

x	vector of strings, e.g. representing table field names.
bFunc	name of a boolean function to be "applied" to each element of *x*.

See Also

hsSqlExOr

Examples

```
## Build SQL query finding records in table t in which all
## of the table fields f1 to f100 are NULL.
sql <- sprintf("SELECT * FROM t WHERE %s",
               hsSqlExAnd(paste("f", 1:100, sep = ""), "isNull"))

## Show the SQL string
sql

## Output (middle part omitted):
# SELECT * FROM t WHERE (TRUE) AND isNull(f1) AND
# isNull(f2) AND isNull(f3) AND ... AND isNull(f100)
```

hsSqlExOr *SQL Expression "OR"*

Description

Creates a boolean expression of the form
`bFunc(x1) OR bFunc(x2) OR ... OR bFunc(xn)`. This function can be used to create SQL queries where many table fields have to be checked in the same way for some criterion (see example).

Usage

```
hsSqlExOr(x, bFunc = "")
```

Arguments

x	vector of strings, e.g. representing table field names.
bFunc	name of a boolean function to be "applied" to each element of *x*.

See Also

hsSqlExAnd

Examples

```
## Build SQL query finding records in table t in which at least
## one of the table fields f1 to f100 is NULL.
sql <- sprintf("SELECT * FROM t WHERE %s",
               hsSqlExOr(paste("f", 1:100, sep = ""), "isNull"))

## Show the SQL string
sql

## Output (middle part omitted):
# SELECT * FROM t WHERE (FALSE) OR isNull(f1) OR
# isNull(f2) OR isNull(f3) OR ... OR isNull(f100)
```

---

hsSqlExTimeCond *SQL Expression: Time Period*

Description

WHERE-condition string in MS Jet SQL syntax filtering for a specific time interval

Usage

```
hsSqlExTimeCond(
  tsField,
  dateFirst = NULL,
  dateLast = NULL,
  inclLast = TRUE,
  sqlDialect = getCurrentSqlDialect(),
  dbg = FALSE
)
```

Arguments

tsField	name of timestamp field
dateFirst	Date object representing the first date of the time interval to be selected.
dateLast	Date object representing the last date of the time interval to be selected.
inclLast	if TRUE, dateLast will be included in result data set, else excluded.
sqlDialect	one of c("mysql", "msaccess")
dbg	if TRUE, debug messages are shown

Value

Condition string in MS Jet SQL syntax to be used in WHERE clause

See Also

hsJetDate

Examples

```
## Not run:
## Condition string to filter field "datetime" for timestamps
## between 21 and 22 of July 2010
from <- as.Date("2011-08-23")
to   <- as.Date("2011-08-24")
cond <- hsSqlExTimeCond("Zeitst", from, to)
cond

## Output:
# TRUE  AND Zeitst >= #08/23/2011 00:00:00#
#       AND Zeitst <= #08/24/2011 00:00:00#

## The condition string may now be used in an SQL query
## to select data from within the time interval.
sql <- sprintf("SELECT * FROM tbl_Hyd WHERE %s", cond)

if (.Platform$OS.type == "windows") {

  res <- hsSqlQuery(xmdb(), sql)
  head(res)
}

## Output:
# Zeitst Q v     H T_Kanal
# 1 2011-08-23 00:00:00 0 0 1.260    19.5
# 2 2011-08-23 00:01:00 0 0 1.259    19.5
# 3 2011-08-23 00:02:00 0 0 1.259    19.5
# 4 2011-08-23 00:03:00 0 0 1.259    19.5
# 5 2011-08-23 00:04:00 0 0 1.260    19.5
# 6 2011-08-23 00:05:00 0 0 1.260    19.5

## End(Not run)
```

`hsSqlExTimeGroup` *SQL Expression: Time Grouping*

Description

Generates SQL code for grouping timestamps by years, months or days

Usage

```
hsSqlExTimeGroup(tbl, tsField, interval, cond = "TRUE")
```

Arguments

`tbl`	name of the table
`tsField`	name of the table field containing the timestamp
`interval`	specifies the time period to group by ("y": years, "m": months, "d": days)
`cond`	additional condition in SQL syntax

Value

Returns SQL code for grouping timestamps by years, months or days

See Also

`hsSqlExTimeCond`

Examples

```
## Show SQL query that gets the number of datasets per
## day ("d") considering the condition "Q > 0"
hsSqlExTimeGroup("myTable", "myTimestamp", "d", "Q > 0")

## Output (reformatted):
## SELECT DateSerial(Year(hsTS), Month(hsTS), Day(hsTS))
##   AS myInterval, Count(*) AS myCount
## FROM (
##     SELECT myTimestamp AS hsTS FROM myTable WHERE Q > 0
## )
## GROUP BY DateSerial(Year(hsTS), Month(hsTS), Day(hsTS))
## ORDER BY DateSerial(Year(hsTS), Month(hsTS), Day(hsTS))
```

hsSqlExTsFields *SQL Expression: Timestamp Info*

Description

Generates SQL code for selecting different parts of the timestamp

Usage

```
hsSqlExTsFields(tsField, extraTsFields = 0:11)
```

Arguments

tsField	name of the table field containing the timestamp
extraTsFields	vector of integers representing different types of time-stamp information: 0 = the timestamp in POSIXct as it is converted to by R from the text representation delivered by MS Access, 1 = Timestamp as character string, 2 = Date, 3 = number of days since 1899-12-30, 4 = number of seconds since midnight of Date 5 = number of minutes since midnight of Date 6:8 = year, month, day 9:11 = hours, minutes, seconds

Value

Returns SQL code for selecting different types of information on the time stamp.

See Also

hsGetTimeSeries

Examples

```
hsSqlExTsFields("myTimestamp", c(6:11))

## Output (re-formatted):
# "Year(myTimestamp) AS myTimestamp_year,
#   Month(myTimestamp) AS myTimestamp_month,
#   Day(myTimestamp) AS myTimestamp_day,
#   Hour(myTimestamp) AS myTimestamp_h,
#   Minute(myTimestamp) AS myTimestamp_m,
#   Second(myTimestamp) AS myTimestamp_s"
```

`hsSqlQuery` *Send SQL Query to Database*

Description

Get data from database requested via an SQL query. This function performs opening of the connection, data retieval via SQL and closing of the connection. If an error occurs the program stops and an error description is shown.

Usage

```
hsSqlQuery(
  mdb,
  sql,
  use2007Driver = NULL,
  dbg = TRUE,
  stopOnError = TRUE,
  DBMSencoding = "",
  ...
)
```

Arguments

`mdb`	full path to MS Access database file (extension ".mdb" or ".accdb") or MS Excel file (extension ".xls" or ".xlsx").
`sql`	SQL query
`use2007Driver`	if TRUE the functions odbcConnectAccess2007 and odbcConnectExcel2007 are used instead of odbcConnectAccess and odbcConnectExcel, respectively
`dbg`	if TRUE (default), debug messages are shown.
`stopOnError`	if TRUE (default), the program stops in case of an error, otherwise a warning is shown and NULL is returned.
`DBMSencoding`	finally passed to `odbcDriverConnect`. Default: "", You may want to use: "UTF-8"
`...`	additional arguments to be passed to `sqlQuery`

Value

On success, a data.frame containing the data that is internally requested by calling the RODBC function sqlQuery and that is provided by the database is returned. On error R stops execution and does not return anything.

See Also

`hsPutTable`, `hsGetTable`

Examples

```
## Not run:
## Get Q time series from table "tbl_Hyd" in example database

if (.Platform$OS.type == "windows") {

  tsQ <- hsSqlQuery(
    xmdb(), "SELECT Zeitst AS t, Q FROM tbl_Hyd WHERE Q > 1.0"
  )

  ## Show the first lines of the resulting data.frame
  head(tsQ)
}

## Output
# t     Q
# 1 2011-08-24 22:27:00 1.061
# 2 2011-08-24 22:28:00 1.091
# 3 2011-08-24 22:29:00 1.115
# 4 2011-08-24 22:30:00 1.092
# 5 2011-08-24 22:31:00 1.086
# 6 2011-08-24 22:32:00 1.074

## End(Not run)
```

---

hsTables *Available tables in database*

Description

Returns a data.frame as returned by sqlTables, containing information on the tables contained in the database. Opening of the database connection, getting the list of tables and closing of the database connection is done within this function.

Usage

```
hsTables(
  mdb,
  excludeSystemTables = grepl("\\.(mdb|accdb)$", mdb),
  namesOnly = TRUE,
  use2007Driver = NULL,
  dbg = FALSE
)
```

Arguments

mdb	full path to MS Access database file (extension ".mdb" or ".accdb") or MS Excel file (extension ".xls" or ".xlsx").

excludeSystemTables
: if TRUE (default), system tables are excluded from the table list, else included.

namesOnly
: if TRUE, only table names are returned. Default: TRUE

use2007Driver
: if TRUE the functions odbcConnectAccess2007 and odbcConnectExcel2007 are used instead of odbcConnectAccess and odbcConnectExcel, respectively

dbg
: if TRUE, debug messages are shown

Value

data.frame with columns *TABLE_CAT*, *TABLE_SCHEM*, *TABLE_NAME*, *TABLE_TYPE*, *REMARKS*, see sqlTables of RODBC package.

See Also

hsFields

Examples

```
## Not run:
## Get names of tables in the example database
## (only on Windows)

if (.Platform$OS.type == "windows") {

  tnames <- hsTables(xmdb(), namesOnly = TRUE)

  ## Exclude system tables by filtering for table names
  ## not starting with '^MSys'
  tNonSys <- grep("^MSys", tnames, invert = TRUE, value = TRUE)

  ## Print the names of the non-system tables.
  cat(paste(tNonSys, "\n"))
}

## Ouput:
# tbl_Hyd
#  tbl_Qua
#  ...

## End(Not run)
```

---

hsTsField *Available timestamp-field(s) in database table*

Description

Available timestamp-field(s) in database table

Usage

```
hsTsField(src, tbl, namesOnly = TRUE, all = FALSE)
```

Arguments

src	source file (MS Access or Excel)
tbl	table name
namesOnly	if TRUE, only the name(s) of the timestamp field(s) is (are) returned, otherwise additional information.
all	if TRUE, all timestamp fields are considiered, otherwise only the first timestamp field.

---

isExcel2003File *Is this an XLS file?*

Description

Is this an XLS file?

Usage

```
isExcel2003File(filepath)
```

Arguments

filepath	(vector of) path(s) to the file(s) to be checked for .xls extension

Value

(vector of) logical.

---

isExcel2007File *Is this an XLSX file?*

Description

Is this an XLSX file?

Usage

```
isExcel2007File(filepath)
```

Arguments

filepath	(vector of) path(s) to the file(s) to be checked for .xlsx extension

Value

(vector of) logical.

`isExcelFile` *Is this an Excel file?*

Description

Is this an Excel file?

Usage

```
isExcelFile(filepath)
```

Arguments

`filepath`	(vector of) path(s) to the file(s) to be checked for .xls or .xlsx extension

Value

(vector of) logical.

`isMySQL` *Is the Given Database of Type MySQL?*

Description

Is the Given Database of Type MySQL?

Usage

```
isMySQL(db, ..., con = NULL)
```

Arguments

`db`	database file (*.mdb, *.accdb, *.xls, *.xlsx) or name of ODBC database
`...`	arguments passed to `hsOpenDb`, e.g. *use2007Driver*
`con`	connection object as returned by `hsOpenDb`, if already available. Default: `NULL`

Value

TRUE if *db* is a MySQL database, else FALSE

keyValuesToSql *List of Key = Value Pairs to SQL Expressions*

Description

List of Key = Value Pairs to SQL Expressions

Usage

```
keyValuesToSql(keyValues, filter, like = filter)
```

Arguments

keyValues	list of key = value pairs
filter	logical. If TRUE the target is an SQL filter expression, otherwise an SQL SET expression.
like	passed to keyValueToSql

Examples

```
keyValues <- list(name = "Peter", birth = as.POSIXct("1999-09-09"))

setCurrentSqlDialect("msaccess")
cat(keyValuesToSql(keyValues, filter = TRUE))
cat(keyValuesToSql(keyValues, filter = TRUE, like = FALSE))
cat(keyValuesToSql(keyValues, filter = FALSE))

setCurrentSqlDialect("mysql")
cat(keyValuesToSql(keyValues, filter = TRUE))
cat(keyValuesToSql(keyValues, filter = TRUE, like = FALSE))
cat(keyValuesToSql(keyValues, filter = FALSE))
```

keyValuesToSqlAssignment *List of Key = Value Pairs to SQL Assignment*

Description

List of Key = Value Pairs to SQL Assignment

Usage

```
keyValuesToSqlAssignment(keyValues)
```

Arguments

keyValues	list of key = value pairs

keyValuesToSqlAssignment2 *Key Values to SQL Assignment (2)*

Description

Key Values to SQL Assignment (2)

Usage

```
keyValuesToSqlAssignment2(keyvalues)
```

Arguments

keyvalues	list of key = value pairs

Value

list with elements *fieldList* and *valueList*

keyValuesToSqlFilter *List of Key = Value Pairs to SQL Filter Expression*

Description

List of Key = Value Pairs to SQL Filter Expression

Usage

```
keyValuesToSqlFilter(keyValues, like = FALSE)
```

Arguments

keyValues	list of key = value pairs
like	if TRUE, the SQL comparison operator will be LIKE instead of =.

keyValueToSql *Generate SQL Filter or Assignment Expression*

Description

Generate SQL Filter or Assignment Expression

Usage

```
keyValueToSql(cname, cvalue, like = TRUE, filter = TRUE)
```

Arguments

cname	field name
cvalue	field value
like	if TRUE, the SQL comparison operator will be LIKE instead of =.
filter	if TRUE an SQL filter expression is returned, otherwise an SQL assignment expression

Value

(vector of) character representing an SQL expression

Examples

```
cat(kwb.db:::keyValueToSql("age", 1))
cat(kwb.db:::keyValueToSql("name", "peter"))
cat(kwb.db:::keyValueToSql("name", "peter", like = FALSE))
```

listValuesToSql *Create SQL Tuples from a List*

Description

Create SQL Tuples from a List

Usage

```
listValuesToSql(x)
```

Arguments

x	a list defining key = value pairs

Value

vector of character strings each of which represents one assignment in x

Examples

```
x <- list(name = "Peter", birthday = as.POSIXct("1981-12-13"))

setCurrentSqlDialect("msaccess")
cat(listValuesToSql(x))

# Note that the representation of a date and time is different in MySQL
setCurrentSqlDialect("mysql")
cat(listValuesToSql(x))
```

---

lookupRecord *Lookup Record*

Description

Looks up a record in a database table and returns the ID of the record or NULL if the record does not exist.

Usage

```
lookupRecord(
  db,
  tableName,
  keyAssignments,
  idField = hsFields(db, tableName)[1],
  dbg = FALSE,
  use2007Driver = NULL
)
```

Arguments

db	full path to MS Access/Excel database or name of ODBC data source
tableName	name of table in which record is to be looked up
keyAssignments	key-value-assignments used to identify the record to be looked up. The assignments are defined in forms of a list, e.g. list(key1 = "value1", key2 = "value2").
idField	name of ID field, default: name of first table field
dbg	if TRUE, debug messages are shown
use2007Driver	passed to isMySQL

merge_relations *Merge Table Schema with Relationship Information*

Description

Merge Table Schema with Relationship Information

Usage

```
merge_relations(schemata, relations)
```

Arguments

schemata	list as returned by read_schemata
relations	data frame as returned by read_relations

openAdequateConnectionOrStop *Open Adequate Connection or Stop*

Description

Open Adequate Connection or Stop

Usage

```
openAdequateConnectionOrStop(
  db,
  use2007Driver = NULL,
  dbg = FALSE,
  DBMSencoding = "",
  ...
)
```

Arguments

db	database name or file
use2007Driver	if TRUE the functions odbcConnectAccess2007 and odbcConnectExcel2007 are used instead of odbcConnectAccess and odbcConnectExcel, respectively
dbg	if TRUE, debug messages are shown
DBMSencoding	finally passed to odbcDriverConnect. Default: "", You may want to use: "UTF-8"
...	further arguments passed to odbcConnectionAccess, odbcConnectionExcel or odbcConnect

printDatabaseSchema *Print Database Schema*

Description

Print Database Schema

Usage

```
printDatabaseSchema(dbSchema)
```

Arguments

dbSchema	database schema as returned by getDatabaseSchema

rcode_DatabaseInterface *R-code for Functions to Access Database Tables*

Description

R-code for Functions to Access Database Tables

Usage

```
rcode_DatabaseInterface(
  db = NULL,
  dbSchema = NULL,
  tableNamePattern = "",
  functionPrefix = "db_",
  pattern = "^",
  dbg = TRUE,
  sql.dbg = TRUE,
  create.create.functions = FALSE
)
```

Arguments

db	database name or file
dbSchema	database schema as returned by getDatabaseSchema
tableNamePattern	pattern matching the names of tables/views for which an accessor function is to be generated
functionPrefix	prefix to be given to each generated function. Default: "db_"

pattern	pattern matching the part of the table name that will be replaced with *functionPrefix*. Default: "^", i.e. the prefix will be appended to the start of the table name
dbg	if TRUE, progress messages are shown during the creation of the interface functions
sql.dbg	if TRUE, SQL strings used to access the database are shown when calling the interface functions
create.create.functions	if TRUE, functions for creating new records in the database tables are created

---

`readSqlCommandsFromFile` *Read SQL Commands from File*

Description

Lines starting with "–" or "#" are ignored. SQL commands must be separated by semicolon and end of line character (\n).

Usage

```
readSqlCommandsFromFile(sqlScript)
```

Arguments

sqlScript	full path to file containing SQL commands

`read_relations` *Read Relations from MSysRelationships Table CSV Export*

Description

Read Relations from MSysRelationships Table CSV Export

Usage

```
read_relations(file)
```

Arguments

file	path to "MSysRelationships.csv" as exported from MS Access by right clicking on the (hidden) MSysRelationships table and selecting "Export->Text file->OK"

Value

data frame with columns `table`, `field`, `refTable`, `refField`

read_schemata *Read Table Schemata from XML Files*

Description

This function reads the *_schema.xml files exported to folder TBL by KWB tool mdbToTxt.exe

Usage

```
read_schemata(path, reduce = TRUE)
```

Arguments

path	path to folder containing xml files
reduce	if TRUE (the default) the returned tables are reduced to relevant fields

renamesToFieldList *List of Renamings to Field Selection String*

Description

Convert a list of renamings to a field selection string that can be used as `fields` argument in a call to `hsGetTable`

Usage

```
renamesToFieldList(
  renames,
  source.in.brackets = TRUE,
  target.in.brackets = TRUE
)
```

Arguments

renames	list of key = value pairs defining renamings from the keys to the values
source.in.brackets	if TRUE (default) the source expressions will be enclosed in square brackets: [source] AS <target>
target.in.brackets	if TRUE (default) the target names will be enclosed in square brackets: <source> AS [target]

renamesToQueries *List of Renamings to SQL Queries*

Description

Convert a list of renamings to a list of SQL queries each of which can be used to select and rename the given fields from a database.

Usage

```
renamesToQueries(renamesList)
```

Arguments

renamesList	list of renaming lists. A renaming list is a list of key = value pairs defining renamings from the keys to the values.

Value

list of character each of which represents an SQL query

runSqlCommandsFromFile *Run SQL Commands from File*

Description

Run SQL Commands from File

Usage

```
runSqlCommandsFromFile(db, sqlFile, ...)
```

Arguments

db	Full path to MS Access mdb file or ODBC database name
sqlFile	full path to file containing Ms Access SQL commands
...	further arguments passed to hsSqlQuery

safeTableName *Safe Table Name*

Description

MS Access: table name enclosed in brackets "[" and "]", else: table name enclosed in backquotes "`"

Usage

```
safeTableName(tableName, sqlDialect = getCurrentSqlDialect(warn = FALSE))
```

Arguments

tableName	name of table to be quoted
sqlDialect	one of c("mysql", "msaccess")

selectFromDb *Select from Database*

Description

Select from Database

Usage

```
selectFromDb(
  tableName,
  fields = "*",
  whereClause = "TRUE",
  odbc,
  dbg = TRUE,
  ...
)
```

Arguments

tableName	name of database table from which to load data
fields	names of fields to be selected
whereClause	SQL WHERE condition string
odbc	database name or file
dbg	if TRUE, debug messages are shown
...	additonal arguments passed to hsSqlQuery

selectFromTable *Select from Table*

Description

Select from Table

Usage

```
selectFromTable(db, tableName, arguments, run = TRUE, ...)
```

Arguments

db	database name or file
tableName	name of table from which to read data
arguments	list with elements select, where, orderBy and further elements starting with select_, where_ and orderBy_, respectively
run	if TRUE (default) the SQL SELECT statement is run otherwise returned as character string
...	further arguments passed to hsSqlQuery

setCurrentDb *Set Current Database*

Description

Set Current Database

Usage

```
setCurrentDb(db)
```

Arguments

db	full path to MS Access database or ODBC database name

setCurrentSqlDialect *Set Current SQL Dialect*

Description

Set Current SQL Dialect

Usage

```
setCurrentSqlDialect(dialectName)
```

Arguments

dialectName	one of "msaccess", "mysql"

sqlForInsert *Generate SQL INSERT statement*

Description

Generate SQL INSERT statement

Usage

```
sqlForInsert(
  tablename,
  fields,
  sqlSource,
  sourceAreValues = !grepl(sqlSource, "^SELECT"),
  ignore = FALSE
)
```

Arguments

tablename	table name
fields	field names, separated by comma
sqlSource	value tupels of form (value1.1, value1.2, value1.3, ...) (value2.1, value2.2, value2.3, ...) ... or SQL SELECT statement providing these tupels
sourceAreValues	if TRUE, *sqlSource* is expected to be an SQL query providing data to be inserted -> no keyword VALUES in generated SQL code
ignore	if TRUE the keyword IGNORE is inserted between INSERT and INTO in the SQL statement -> no error will be given if data to insert already exists

sqlForInsertDataFrame *Generate SQL INSERT Statement*

Description

Generate SQL INSERT statement to insert values in a data frame

Usage

```
sqlForInsertDataFrame(tablename, dataFrame, ignore = FALSE, ...)
```

Arguments

tablename	table name
dataFrame	data frame with column names representing table field names
ignore	if TRUE the keyword IGNORE is inserted between INSERT and INTO in the SQL statement -> no error will be given if data to insert already exists
...	further arguments passed to sqlForInsert

sqlForInsertFromSelect *SQL for INSERT FROM SELECT*

Description

Generate SQL INSERT statement of the form INSERT INTO target.table (fields) SELECT fields FROM source.table

Usage

```
sqlForInsertFromSelect(target.table, source.table, fields)
```

Arguments

target.table	name of target table
source.table	name of source table or SQL providing source data
fields	vector of character with field names

sqlForInsertIgnoreInMsAccess *SQL for "INSERT IGNORE" in MS Access*

Description

Returns SQL for inserting all records from table.source that are not yet contained in table.target into table.target

Usage

```
sqlForInsertIgnoreInMsAccess(db, table.source, table.target, uniqueFields = NA)
```

Arguments

db	database name or file
table.source	name of source table
table.target	name of target table
uniqueFields	names of unique fields

sqlForSelect *Generate SQL SELECT statement#'*

Description

Generate SQL SELECT statement#'

Usage

```
sqlForSelect(
  tablename,
  fields = "*",
  whereClause = "TRUE",
  groupBy = "",
  orderBy = "",
  sqlDialect = getCurrentSqlDialect()
)
```

Arguments

tablename	table name
fields	expression to select fields; field names are separated by comma and alias names may be used, just as SQL accepts, e.g.: "tstamp as myDateTime, parVal as myValue"; Default: "*"
whereClause	where condition; Default: "TRUE"
groupBy	GROUP BY-clause, Default: "" (no grouping)
orderBy	ORDER BY-clause, Default: "" (no sorting of results)
sqlDialect	one of c("mysql", "msaccess")

---

sqlForSelectByKey *Generate SQL SELECT Statement*

Description

Generate SQL SELECT statement (key field values instead of where clause)

Usage

```
sqlForSelectByKey(tablename, fields = "*", keyValues = NULL)
```

Arguments

tablename	table name
fields	expression to select fields; field names are separated by comma and alias names may be used, just as SQL accepts, e.g.: "tstamp as myDateTime, parVal as myValue"; Default: "*"
keyValues	list of "key=value" pairs with the keys being valid field names of *table*

---

sqlForUpdate *Generate SQL UPDATE Statement*

Description

Generate SQL UPDATE Statement

Usage

```
sqlForUpdate(tablename, keyValues, whereClause, ignore = FALSE)
```

Arguments

tablename	table name
keyValues	assignments as list of *key*=*value* pairs with the keys representing valid fields of table *tablename*
whereClause	where condition
ignore	if TRUE the keyword IGNORE is inserted between UPDATE and INTO in the SQL statement -> no error will be given if updating fails, e.g. because of key constraints

---

sqlFullLeftJoin *Merge SQL Queries to One Query That Performs a Full Left Join*

Description

Merge SQL Queries to One Query That Performs a Full Left Join

Usage

```
sqlFullLeftJoin(sqls, key)
```

Arguments

sqls	list of SQL queries each of which is expected to contain an attribute `alias` giving the alias name for the query
key	name of the primary key field, being selected in each SQL query contained in `sqls`

Value

vector of character of length one representing the result of "left join"-ing all sub-queries given in `sqls`

Examples

```
sql <- sqlFullLeftJoin(key = "id", list(
  structure("SELECT id, field_1 from table_1", alias = "t1"),
  structure("SELECT id, field_2, field_3 from table_2", alias = "t2"),
  structure("SELECT id, field_4 from table_3", alias = "t3")
))

cat(sql)
```

sqlJoinExpression *Create an SQL JOIN Expression*

Description

Create an SQL JOIN Expression

Usage

```
sqlJoinExpression(left, right, condition, type = "INNER")
```

Arguments

left	left part of JOIN (e.g. table name)
right	right part of JOIN (e.g. table name)
condition	condition
type	one of c("LEFT", "RIGHT", "INNER")

sqlLeftJoin *Generate SQL for LEFT JOIN*

Description

Generate SQL for LEFT JOIN

Usage

```
sqlLeftJoin(sqlSource, tablesAndIDs, fields = "*")
```

Arguments

sqlSource	SQL of subquery that provides the "base" table on the left
tablesAndIDs	named character vector with the names representing the names of the tables to be joined and the values representing the ID fields of these tables
fields	fields to be selected

sqlLeftJoinBody *Generate (Base Part of) SQL for LEFT JOIN*

Description

Generate (Base Part of) SQL for LEFT JOIN

Usage

```
sqlLeftJoinBody(
  leftSql,
  rightTable,
  id,
  idLeft = id,
  useAlias = (id != idLeft),
  aliasName = "tbase"
)
```

Arguments

leftSql	SQL of subquery that provides the "base" table on the left
rightTable	name of "right" table
id	name of ID field of "right" table (must correspond to a field returned by *leftSql*)
idLeft	name of ID field of "left" table
useAlias	if TRUE, the alias given in *aliasName* is given to the subquery *leftSql*. Default: FALSE
aliasName	alias name to be used if *useAlias* is TRUE. Default: "tbase"

sqlLeftJoinExpression *Create an SQL LEFT JOIN Expression*

Description

Create an SQL LEFT JOIN Expression

Usage

```
sqlLeftJoinExpression(left, right, condition)
```

Arguments

left	left part of JOIN (e.g. table name)
right	right part of JOIN (e.g. table name)
condition	condition

xmdb *Path to example database*

Description

Returns full path to MS Access example database

Usage

```
xmdb()
```

Index