Package: dbmisc (via r-universe)

September 4, 2024

Type Package

Title Tools for working with SQLite in R

Version 0.3

Date 2021-03-11

Author Sebastian Kranz

Maintainer Sebastian Kranz <sebastian.kranz@uni-ulm.de>

Description Tools for working with SQLite in R, in particular support
for simple YAML schemas.

License GPL >= 2.0

LazyData TRUE

Depends DBI, restorepoint, yaml, stringtools, glue

RoxygenNote 7.1.1

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

RemoteUrl https://github.com/skranz/dbmisc

RemoteRef master

RemoteSha aef87129585de20453627f7246effebf00fac596

Contents

convert.db.to.r *Convert data from a database table to R format*

Description

Convert data from a database table to R format

Usage

```
convert.db.to.r(
  vals,
  rclass = schema$rclass,
  schema = NULL,
  as.data.frame = is.data.frame(vals),
  null.as.na = TRUE,
  origin = "1970-01-01"
)
```

Arguments

`vals`	the values loaded from the database table
`rclass`	the r class of the table columns, is extracted from schema
`schema`	a table schema that can be used to convert values
`null.as.na`	shall NULL values be converted to NA values?
`origin`	the origin date for DATE and DATETIME conversion

`convert.r.to.db` *Convert data from a database table to R format*

Description

Convert data from a database table to R format

Usage

```
convert.r.to.db(
  vals,
  rclass = schema$rclass,
  schema = NULL,
  null.as.na = TRUE,
  origin = "1970-01-01",
  add.missing = TRUE
)
```

Arguments

`vals`	the values loaded from the database table
`rclass`	the r class of the table columns, is extracted from schema
`schema`	a table schema that can be used to convert values
`null.as.na`	shall NULL values be converted to NA values?
`origin`	the origin date for DATE and DATETIME conversion

`dbConnectSQLiteWithSchema` *Creates a connection to an SQLite database and sets the specified schema*

Description

The schema is added as attribute to the connection object and is automatically used by `dbInsert`, `dbGet`, and `dbUpdate`.

Usage

```
dbConnectSQLiteWithSchema(
  dbname,
  schema.file = paste0(tools::file_path_sans_ext(dbname), ".yaml"),
  schema = load.and.init.schemas(schema.file)
)
```

Arguments

`dbname`	Filename of the SQLite database
`schema.file`	YAML file that contains the database schema. By default it is assumed to be in the same folder as the database with the same name but the extension ".yaml"
`schema`	If you already loaded a schema file manually with `load.and.init.schemas`, you can also provide it here instead of specifying a schema.file.

`dbCreateSchemaTable` *Create database table and possible indices from a simple yaml schema*

Description

Create database table and possible indices from a simple yaml schema

Usage

```
dbCreateSchemaTable(
  db,
  table,
  schema = schemas[[table]],
  schemas = get.db.schemas(db),
  schema.yaml = NULL,
  schema.file = NULL,
  overwrite = update,
  silent = FALSE,
  update = TRUE,
  verbose = 1
)
```

Arguments

`db`	dbi database connection
`schemas`	schemas as R list
`schema.yaml`	alternatively a schema as yaml text
`schema.file`	alternatively a file name of a schema yaml file
`overwrite`	shall existing tables be overwritten?
`silent`	if TRUE don't show messages
`update`	shall old data be copied from existing tables?

dbCreateSchemaTables — *Create or update database tables and possible indices from a simple yaml schema*

Description

Create or update database tables and possible indices from a simple yaml schema

Usage

```
dbCreateSchemaTables(
  db,
  schemas = get.db.schemas(db),
  schema.yaml = NULL,
  schema.file = NULL,
  overwrite = update,
  silent = FALSE,
  update = TRUE,
  verbose = 1
)
```

Arguments

`db`	dbi database connection
`schemas`	schemas as R list
`schema.yaml`	alternatively a schema as yaml text
`schema.file`	alternatively a file name of a schema yaml file
`overwrite`	shall existing tables be overwritten?
`silent`	if TRUE don't show messages
`update`	if TRUE (default) copy old data from existing tables.

dbCreateSQLiteFromSchema — *Create or update a SQLite database from a schema file*

Description

Create or update a SQLite database from a schema file

Usage

```
dbCreateSQLiteFromSchema(
  schema.file,
  schema.dir = dirname(schema.file),
  db.name = NULL,
  db.dir = schema.dir,
  update = TRUE,
  verbose = 1
)
```

Arguments

`schema.file`	the dbmisc schema file in yaml format
`schema.dir`	the directory of the schema file (if schema.file does not contain a path)
`db.name`	the name of the database file
`db.dir`	the directory of the database file, by default the schema directory
`update`	if TRUE copy and update the existing data in the tables. If FALSE just generate empty tables.
`verbose`	if 0 don't show what is done. If 1 or larger show most of the run SQL commands.

`dbDelete` *Delete row(s) from table*

Description

Delete row(s) from table

Usage

```
dbDelete(
  db,
  table,
  params,
  sql = NULL,
  run = TRUE,
  log.dir = NULL,
  do.log = !is.null(log.dir),
  user = NA,
  where.in = FALSE
)
```

Arguments

`db`	dbi database connection
`table`	name of the table
`params`	named list of values for key fields that identify the rows to be deleted
`sql`	optional a parameterized sql string
`run`	if FALSE only return parametrized SQL string

`dbGet` *Get rows from a table*

Description

Get rows from a table

Usage

```
dbGet(
  db,
  table = NULL,
  params = NULL,
  sql = NULL,
  fields = NULL,
  joinby = NULL,
  jointype = c("inner", "left", "right")[1],
  run = TRUE,
  schema = if (length(table) == 1) schemas[[table]] else NULL,
  schemas = get.db.schemas(db),
  rclass = schema$rclass,
  convert = !is.null(rclass),
  convert.param = FALSE,
  orderby = NULL,
  null.as.na = TRUE,
  origin = "1970-01-01",
  where.in = FALSE,
  where.sql = NULL,
  empty.as.null = FALSE,
  n = -1
)
```

Arguments

`db`	dbi database connection
`table`	name of the table. If you specify more than one table the later tables will be joined. You then should specify the `joinby` argument and possible the `fields` argument if you want to select fields also from the later tables.

`params`	named list of values for key fields. If you don't use a custom SQL statement the list will be used to construct a WHERE clause. E.g. `params = list(age=30,gender="male")` would be translated to the WHERE clause `WHERE age = 30 AND gender="male"`. If you want to match several values, e.g. `params = list(age = c(30,40))` you need to set the argument `where.in = TRUE` to construct a correct WHERE clause.
`sql`	optional a parameterized custom sql string Can contain parameters passed with the `param` arguments. E.g. if you have `param = list(myname="Seb")` you could use `myname` in an SQL statement as follows: select * from mytable where name = :myname To avoid SQL injection you should provide all values that can be provided by a user as such parameters or make sure that you escape them.
`fields`	If not NULL can be used to specify fields that shall be selected as character. For joined tables, you must enter fields in the format "tablename.field". E.g. `fields = "*, table2.myfield"` would select all columns from the first table and the column `myfield` from the joined 2nd table.
`joinby`	If you specify more than one table the later tables shall be joined by the variables specified in `joinby` with the first table. For more complicated joins where the names of the join variables differ you have to write custom SQL with the `sql` argument instead.
`jointype`	The type of the join if you specify a `joinby` argument. Default is "inner" but can also be set to "left" or "right"
`run`	if FALSE only return parametrized SQL string
`schema`	a table schema that can be used to convert values
`rclass`	the r class of the table columns, is extracted from schema
`convert`	if rclass is given shall results automatically be converted to these classes?
`orderby`	names of columns the results shall be ordered by as character vector. Add "DESC" or "ASC" after column name to sort descending or ascending. Example: `orderby = c("pos DESC","hp ASC")`
`null.as.na`	shall NULL values be converted to NA values?
`origin`	the origin date for DATE and DATETIME conversion
`where.in`	Set TRUE if your params contain sets and therefore a WHERE IN clause shall be generated.
`where.sql`	An optional SQL code just for the WHERE clause. Can be used if some parameters will be checked with inequality.
`empty.as.null`	if TRUE return just NULL if the query returns zero rows.
`n`	The maximum number of rows that shall be fetched. If `n=-1` (DEFAULT) fetch all rows.

`dbGetMemoise` *Get results from a database like dbGet put buffer the results in memory*

Description

If the function is called again with the same parameter check if the something was changed in the database inbetween by looking at the time stamp of the log file. If there were no changes restore the values from memory. If there were changes load data again from database.

Usage

```
dbGetMemoise(
  db,
  table,
  params = NULL,
  schema = schemas[[table]],
  schemas = get.db.schemas(db),
  log.dir = NULL,
  refetch.if.changed = !is.null(log.dir),
  empty.as.null = FALSE,
  ...
)
```

Details

If refetch.if.changed = FALSE (default if no log.dir is provided), always use the data from memory.

`dbInsert` *Insert row(s) into table*

Description

Insert row(s) into table

Usage

```
dbInsert(
  db,
  table = NULL,
  vals,
  schema = schemas[[table]],
  schemas = get.db.schemas(db),
  sql = NULL,
  run = TRUE,
  mode = c("insert", "replace")[1],
  add.missing.cols = TRUE,
  rclass = schema$rclass,
  convert = !is.null(rclass),
  primary.key = schema$primary_key,
  get.key = FALSE,
  null.as.na = TRUE,
  log.dir = NULL,
  do.log = !is.null(log.dir),
  user = NA
)
```

Arguments

`db`	dbi database connection
`table`	name of the table
`vals`	named list of values to be inserted
`schema`	a table schema that can be used to convert values
`sql`	optional a parameterized sql string
`run`	if FALSE only return parametrized SQL string
`mode`	"insert" or "replace", should have no effect so far
`add.missing.cols`	if TRUE (default) and a schema is provided than automatically add database columns that are missing in `vals` and set them NA.
`rclass`	the r class of the table columns, is extracted from schema
`convert`	if rclass is given shall results automatically be converted to these classes?
`primary.key`	name of the primary key column (if the table has one)
`get.key`	if TRUE return the created primary key value

`dbTableCols` *Get a data frame with column information for a database table*

Description

Get a data frame with column information for a database table

Usage

```
dbTableCols(db, table)
```

Arguments

`db`	dbi database connection
`table`	name of the table

dbUpdate *Update a row in a database table*

Description

Update a row in a database table

Usage

```
dbUpdate(
  db,
  table,
  vals,
  where = NULL,
  schema = schemas[[table]],
  schemas = get.db.schemas(db),
  sql = NULL,
  run = TRUE,
  rclass = schema$rclass,
  convert = !is.null(rclass),
  null.as.na = TRUE,
  log.dir = NULL,
  do.log = !is.null(log.dir),
  user = NA,
  where.in = FALSE
)
```

Arguments

`db`	dbi database connection
`table`	name of the table
`vals`	named list of values to be inserted
`where`	named list that specifies the keys where to update
`schema`	a schema as R list, can be used to automatically convert types
`sql`	optional a parameterized sql string
`run`	if FALSE only return parametrized SQL string
`rclass`	the r class of the table columns, is extracted from schema
`convert`	if rclass is given shall results automatically be converted to these classes?
`null.as.na`	shall NULL values be converted to NA values?

`empty.df.from.schema` *Creates an example data frame from a database schema table using provided column values and default values specified in schema*

Description

Creates an example data frame from a database schema table using provided column values and default values specified in schema

Usage

```
empty.df.from.schema(.schema, .nrows = 1, ..., .use.defaults = TRUE)
```

`empty.row.from.schema` *Creates an example row from a database schema table using provided column values and default values specified in schema*

Description

Creates an example row from a database schema table using provided column values and default values specified in schema

Usage

```
empty.row.from.schema(.schema, ..., .use.defaults = TRUE)
```

`get.db.schema` *Extract schemas from a data base connection*

Description

Extract schemas from a data base connection

Usage

```
get.db.schema(db, warn.null = TRUE)
```

`init.schema` *Init a schema by parsing table definition and store info in easy accessibale R format*

Description

Create rclasses of each column and primary keys

Usage

```
init.schema(schema, name = NULL)
```

Arguments

`schema`	the table schema as an R list
`name`	of the table

`load.and.init.schemas` *Load and init database table schemas from yaml file*

Description

Load and init database table schemas from yaml file

Usage

```
load.and.init.schemas(file = NULL, yaml = readLines(file, warn = FALSE))
```

Arguments

`file`	file name
`yaml`	yaml as text

`logDBcommand`	*log a command that changes a database*

Description

log a command that changes a database

Usage

```
logDBcommand(
  type,
  sql = "",
  user = "NA",
  log.dir = NULL,
  table = NULL,
  do.log = TRUE
)
```

`schema.r.classes`	*Get a vector of R classes of the database columns described in a schema*

Description

Get a vector of R classes of the database columns described in a schema

Usage

```
schema.r.classes(schema)
```

Arguments

`schema`	the schema

`schema.template` *Create an example schema from a list of R objects*

Description

The output is shown per cat and copied to the clipboard. It can be used as a template for the .yaml schema file

Usage

```
schema.template(li, name = "mytable", toClipboard = TRUE)
```

Arguments

`li`	The R list for which the schema shall be created
`name`	optional a name of the table
`toCliboard`	shall the created text be copied to the clipboard

`set.db.schema` *Set schemas as hidden attribute to a data base connection db*

Description

Set schemas as hidden attribute to a data base connection db

Usage

```
set.db.schema(db, schemas = NULL, schema.file = NULL)
```

`sql.where.code` *Create a parametrized or escaped SQL WHERE clause from the provided parameters.*

Description

Create a parametrized or escaped SQL WHERE clause from the provided parameters.

Usage

```
sql.where.code(
  db = NULL,
  params,
  where.in = FALSE,
  parametrized = !where.in,
  add.where = TRUE
)
```

Arguments

`db`	a database connection needed for correct escaping via glue_sql
`params`	a list of parameters assume that db fields have the same name and have to be equal to provided values.
`where.in`	Set true TRUE a member of params can be vector and return all rows that match an element. By default FALSE to generate more compact code.
`add.where`	If TRUE start with WHERE
`paramertrized`	shall the generated code use SQL parameters.

`to.db.date` *Convert an R object to a date object that can be used in a WHERE clause.*

Description

Convert an R object to a date object that can be used in a WHERE clause.

Usage

```
to.db.date(val, origin = "1970-01-01")
```

`to.db.datetime` *Convert an R object to a datetime object that can be used in a WHERE clause.*

Description

Convert an R object to a datetime object that can be used in a WHERE clause.

Usage

```
to.db.datetime(val, origin = "1970-01-01")
```

Index