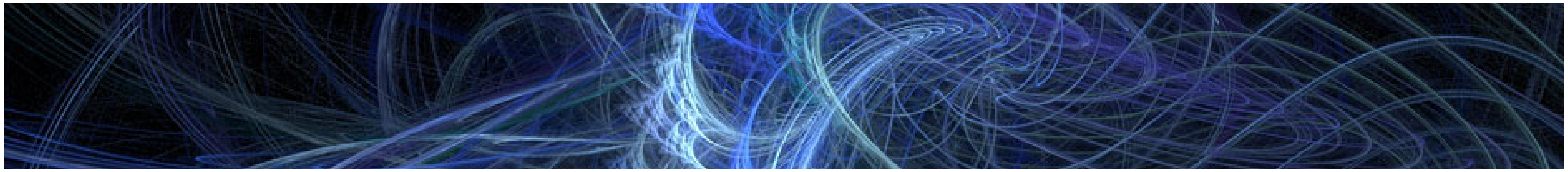
inside-R

A Community Site for R – Sponsored by Revolution Analytics



sqlSave {RODBC}

Write a Data Frame to a Table in an ODBC Database

Package: RODBC
Version: 1.3-13

Description

Write or update a table in an ODBC database.

Usage

```
sqlSave(channel, dat, tablename = NULL, append = FALSE,
        rownames = TRUE, colnames = FALSE, verbose = FALSE,
        safer = TRUE, addPK = FALSE, typeInfo, varTypes,
        fast = TRUE, test = FALSE, nastring = NULL)

sqlUpdate(channel, dat, tablename = NULL, index = NULL,
          verbose = FALSE, test = FALSE, nastring = NULL,
          fast = TRUE)
```

Arguments

`channel`
: connection handle returned by `odbcConnect`.

`dat`
: a data frame.

`tablename`
: character: a database table name accessible from the connected DSN. If missing, the name of `dat`.

`index`
: character. Name(s) of index column(s) to be used.

`append`
: logical. Should data be appended to an existing table?

`rownames`
: either logical or character. If logical, save the row names as the first column `rownames` in the table? If character, the column name under which to save the rownames.

`colnames`
: logical: save column names as the first row of table?

`verbose`
: display statements as they are sent to the server?

`safer`
: logical. If true, create a non-existing table but only allow appends to an existing table. If false, allow `sqlSave` to attempt to delete all the rows of an existing table, or to drop it.

`addPK`
: logical. Should rownames (if included) be specified as a primary key?

`typeInfo`
: optional list of DBMS datatypes. Should have elements named `"character"`, `"double"` and `"integer"`.

`varTypes`
: an optional named character vector giving the DBMSs datatypes to be used for some (or all) of the

columns if a table is to be created.

`fast`
: logical. If false, write data a row at a time. If true, use a parametrized `INSERT INTO` or `UPDATE` query to write all the data in one operation.

`test`
: logical: if `TRUE` show what would be done, only.

`nastring`
: optional character string to be used for writing `NA`s to the database. See ‘Details’.

Details

`sqlSave` saves the data frame `dat` in the table `tablename`. If the table exists and has the appropriate structure it is used, or else it is created anew. If a new table is created, column names are remapped by removing any characters which are not alphanumeric or `_`, and the types are selected by consulting arguments `varTypes` and `typeInfo`, then looking the driver up in the database used by `getSqlTypeInfo` or failing that by interrogating `sqlTypeInfo`.

If `rownames = TRUE` the first column of the table will be the row labels with colname `rowname`: `rownames` can also be a string giving the desired column name (see ‘Examples’). If `colnames` is true, the column names are copied into row 1. This is intended for cases where case conversion alters the original column names and it is desired that they are retained. Note that there are drawbacks to this approach: it presupposes that the rows will be returned in the correct order; not always valid. It will also cause numeric columns to be returned as factors.

Argument `addPK = TRUE` causes the row names to be marked as a primary key. This is usually a good idea, and may allow database updates to be done. However, the ODBC drivers for some DBMSs (e.g. Access) do not support primary keys, and earlier versions of the PostgreSQL ODBC driver generated internal memory corruption if this option is used. `sqlUpdate` updates the table where the rows already exist. Data frame `dat` should contain columns with names that map to (some of) the columns in the table. It also needs to contain the column(s) specified by `index` which together identify the rows to be updated. If `index = NULL`, the function tries to identify such columns. First it looks for a primary key for the table, then for the column(s) that the database regards as the optimal for defining a row uniquely (these are returned by `sqlColumns(special = TRUE)`: if this returns a pseudo-column it cannot be used as we do not have values for the rows to be changed). Finally, the row names are used if they are stored as column "rownames" in the table. When fast = TRUE, NAs are always written as SQL nulls in the database, and this is also the case if fast = FALSE and nastring = NULL (its default value). Otherwise nastring gives the character string to be sent to the driver when NAs are encountered: for all but the simplest applications it will be better to prepare a data frame with non-null missing values already substituted.

If fast = FALSE all data are sent as character strings. If fast = TRUE, integer and double vectors are sent as types SQL_C_SLONG and SQL_C_DOUBLE respectively. Some drivers seem to require fast = FALSE to send other types, e.g. datetime. SQLite's approach is to use the data to determine how it is stored, and this does not work well with fast = TRUE.

If tablename contains . and neither catalog nor schema is supplied, an attempt is made to interpret <var>qualifier</var>.<var>table</var> names as table <var>table</var> in schema <var>qualifier</var> (and for MySQL ‘schema’ means ‘database’). (This can be suppressed by opening the connection with interpretDot = FALSE.)

Values

1 invisibly for success (and failures cause errors).

Warning

sqlSave(safer = FALSE) uses the ‘great white shark’ method of testing tables (bite it and see). The logic will unceremoniously DROP the table and create it anew with its own choice of column types in its attempt to find a writable solution. test = TRUE will not necessarily predict this behaviour. Attempting to write indexed columns or writing to pseudo-columns are less obvious causes of failed writes followed by a DROP. If your table structure is precious it is up to you back it up.

See Also

sqlFetch, sqlQuery, odbcConnect, odbcGetInfo

Examples

```
## Not run:
```

```
channel <- odbcConnect("test")
sqlSave(channel, USArrests, rownames = "state", addPK=TRUE)
sqlFetch(channel, "USArrests", rownames = "state") # get the lot
foo <- cbind(state=row.names(USArrests), USArrests)[1:3, c(1,3)]
foo[1,2] <- 222
sqlUpdate(channel, foo, "USArrests")
sqlFetch(channel, "USArrests", rownames = "state", max = 5)
sqlDrop(channel, "USArrests")
close(channel)
## End(Not run)
```

Author(s)

Michael Lapsley and Brian Ripley