Package 'RPostgreSQL'

February 19, 2015

Version 0.4

Date $Date: 2013-03-27 15:32:53 +0900 (Wed, 27 Mar 2013) $

Title R interface to the PostgreSQL database system

Author Joe Conway, Dirk Eddelbuettel, Tomoaki Nishiyama, Sameer Kumar Prayaga (during 2008), Neil Tiffin

Maintainer Tomoaki Nishiyama <tomoakin@staff.kanazawa-u.ac.jp>

Description Database interface and PostgreSQL driver for R This package provides a Database Interface (DBI) compliant driver for R to access PostgreSQL database systems.
In order to build and install this package from source, PostgreSQL itself must be present your system to provide PostgreSQL functionality via its libraries and header files. These files are provided as postgresql-devel package under some Linux distributions.
On Microsoft Windows system the attached libpq library source will be used.
A wiki and issue tracking system for the package are available at Google Code at https://code.google.com/p/rpostgresql/ .

LazyLoad true

Depends R (>= 2.9.0), methods, DBI (>= 0.1-4)

License GPL-2 | file LICENSE

Copyright Authors listed above, PostgreSQL Global Development Group, and The Regents of the University of California

Collate S4R.R zzz.R PostgreSQLSupport.R dbObjectId.R PostgreSQL.R

URL https://code.google.com/p/rpostgresql/, http://www.stat.bell-labs.com/RS-DBI, http://www.postgresql.org

NeedsCompilation yes

Repository CRAN

Date/Publication 2013-03-27 09:34:11

R topics documented:

`dbApply` *Apply R/S-Plus functions to remote groups of DBMS rows (experimental)*

Description

Applies R/S-Plus functions to groups of remote DBMS rows without bringing an entire result set all at once. The result set is expected to be sorted by the grouping field.

Details

`dbApply` This generic is meant to handle somewhat gracefully(?) large amounts of data from the DBMS by bringing into R manageable chunks; the idea is that the data from individual groups can be handled by R, but not all the groups at the same time.

Currently, only the `PostgreSQL` driver implements a method (see the helper function `postgresqlDBApply`) for this generic function.

Value

A list with as many elements as there were groups in the result set.

See Also

PostgreSQL postgresqlDBApply dbSendQuery fetch

Examples

```
## Not run:
## compute quantiles for each network agent
con <- dbConnect(PostgreSQL(), user= "user", password="passwd", dbname="sample")
rs <- dbSendQuery(con,
             "select Agent, ip_addr, DATA from pseudo_data order by Agent")
out <- dbApply(rs, INDEX = "Agent",
        FUN = function(x, grp) quantile(x$DATA, names=FALSE))

## End(Not run)
```

---

dbApply-methods *Apply R/S-Plus functions to remote groups of DBMS rows (experimental)*

Description

Applies R/S-Plus functions to groups of remote DBMS rows without bringing an entire result set all at once. The result set is expected to be sorted by the grouping field.

Methods

res a PostgreSQL result set (see dbSendQuery).

... any additional arguments to be passed to FUN.

References

See the Database Interface definition document DBI.pdf in the base directory of this package or http://stat.bell-labs.com/RS-DBI.

See Also

PostgreSQL postgresqlDBApply dbSendQuery fetch

Examples

```
## Not run:
## compute quanitiles for each network agent
con <- dbConnect(PostgreSQL(), user="user", password="passwd",dbname="dbname")
rs <- dbSendQuery(con,
             "select Agent, ip_addr, DATA from pseudo_data order by Agent")
out <- dbApply(rs, INDEX = "Agent",
        FUN = function(x, grp) quantile(x$DATA, names=FALSE))

## End(Not run)
```

---

dbCallProc-methods *Call an SQL stored procedure*

Description

Not yet implemented.

Methods

conn a PostgreSQLConnection object.

... additional arguments are passed to the implementing method.

References

See the Database Interface definition document DBI.pdf in the base directory of this package or http://stat.bell-labs.com/RS-DBI.

See Also

PostgreSQL, dbConnect, dbSendQuery, dbGetQuery, fetch, dbCommit, dbGetInfo, dbReadTable.

---

dbCommit-methods *DBMS Transaction Management*

Description

Commits or roll backs the current transaction in an PostgreSQL connection. dbCommit and dbRollback commit and rollback the transaction, respectively.

Methods

conn a PostgreSQLConnection object, as produced by the function dbConnect.

... currently unused.

References

See the Database Interface definition document DBI.pdf in the base directory of this package or http://stat.bell-labs.com/RS-DBI.

See Also

PostgreSQL, dbConnect, dbSendQuery, dbGetQuery, fetch, dbCommit, dbGetInfo, dbReadTable.

Examples

```
## Not run:
drv <- dbDriver("PostgreSQL")
con <- dbConnect(drv, dbname="postgres")
dbGetQuery(con, "select count(*) from sales")

dbGetQuery(con, "BEGIN TRANSACTION")
rs <- dbSendQuery(con,
      "Delete * from sales as p where p.cost>10")
if(dbGetInfo(rs, what = "rowsAffected") > 250){
  warning("Rolling back transaction")
  dbRollback(con)
}else{
  dbCommit(con)
}

dbGetQuery(con, "select count(*) from sales")
dbDisconnect(con)

## End(Not run)
```

---

dbConnect-methods *Create a connection object to an PostgreSQL DBMS*

Description

These methods are straight-forward implementations of the corresponding generic functions.

Methods

drv an object of class PostgreSQLDriver, or the character string "PostgreSQL" or an PostgreSQLConnection.

conn an PostgreSQLConnection object as produced by dbConnect.

host Name or the numeric IPaddress of the host to connect to. If address is specified, it should be in the standard IPv4 address format, e.g., 172.28.40.9 or if your machine supports IPv6, you can also use those addresses. The default is to connect to localhost.

dbname The database name. Defaults to 'template1' i.e if this argument is not specified, it is taken as dbname="template1".

user PostgreSQL user name to connect as. Defaults to be the same as the operating system name of the user running the application.

password Password to be used if the server demands password authentication.

port Port number to connect to at the server host, or socket file name extension for Unix-domain connections.

tty Ignored (formerly, this specified where to send server debug output).

options Command-line options to be sent to the server

forceISOdate logical if the communication of date (time stamp) from PostgreSQL is forced to ISO style at conection.

Side Effects

A connection between R/S-Plus and an PostgreSQL server is established. The current implementation supports up to 100 simultaneous connections.

References

See the Database Interface definition document `DBI.pdf` in the base directory of this package or http://stat.bell-labs.com/RS-DBI.

See Also

`PostgreSQL`, `dbConnect`, `dbSendQuery`, `dbGetQuery`, `fetch`, `dbCommit`, `dbGetInfo`, `dbReadTable`.

Examples

```
## Not run:
# create an PostgreSQL instance and create one connection.
drv <- dbDriver("PostgreSQL")

# open the connection using user, passsword, etc., as
con <- dbConnect(drv, dbname = "postgres")

df <- dbGetQuery(con, statement = paste(
                      "SELECT itemCode, itemCost, itemProfit",
                      "FROM sales",
                      "SORT BY itemName"));
# Run an SQL statement by creating first a resultSet object
rs <- dbSendQuery(con, statement = paste(
                      "SELECT itemCode, itemCost, itemProfit",
                      "FROM sales",
                      "SORT BY itemName"));

# we now fetch records from the resultSet into a data.frame
df <- fetch(rs, n = -1)   # extract all rows
dim(df)

## End(Not run)
```

dbDataType-methods *Determine the SQL Data Type of an S object*

Description

This method is a straight-forward implementation of the corresponding generic function.

Methods

dbObj any PostgreSQLObject object, e.g., PostgreSQLDriver, PostgreSQLConnection, PostgreSQLResult.

obj R/S-Plus object whose SQL type we want to determine.

... any other parameters that individual methods may need.

References

See the Database Interface definition document DBI.pdf in the base directory of this package or http://stat.bell-labs.com/RS-DBI.

See Also

isSQLKeyword make.db.names

Examples

```
## Not run:
data(quakes)
drv <- dbDriver("PostgreSQL")
sql.type <- dbDataType(drv, quakes)

## End(Not run)
```

dbDriver-methods *PostgreSQL implementation of the Database Interface (DBI) classes and drivers*

Description

PostgreSQL driver initialization and closing

Methods

drvName character name of the driver to instantiate.

drv an object that inherits from PostgreSQLDriver as created by dbDriver.

max.con optional integer requesting the maximum number of simultanous connections (may be up to 100).

fetch.default.rec default number of records to retrieve per fetch. Default is 500. This may be overridden in calls to fetch with the n= argument.

force.reload optional logical used to force re-loading or recomputing the size of the connection table. Default is FALSE.

... currently unused.

References

See the Database Interface definition document DBI.pdf in the base directory of this package or http://stat.bell-labs.com/RS-DBI.

See Also

PostgreSQL, dbConnect, dbSendQuery, dbGetQuery, fetch, dbCommit, dbGetInfo, dbListTables, dbReadTable.

Examples

```
## Not run:

# create an PostgreSQL instance and set 10000 of rows per fetch.
library(RPostgreSQL)
drv <- dbDriver("PostgreSQL", fetch.default.records=10000)

# Connecting to PostgreSQL with the specified parameters
con <- dbConnect(drv,user="usrname",password="passwd",dbname="postgres")

# Running the query to obtain the resultset
rs <- dbSendQuery(con, "select * from cities where population > 5000")

# fetch records into a dataframe.
# n = 50 fetched fifty records
df <- fetch(rs, n = 50)
# n = -1 fetches all the remaining records available
df2 <- fetch(rs, n = -1)

# Clearing the result set
dbClearResult(rs)

#This returns a character vector (possibly of zero-length)
# table names available on the con connection.
dbListTables(con)

## End(Not run)
```

dbGetInfo-methods *Database interface meta-data*

Description

These methods are straight-forward implementations of the corresponding generic functions.

Methods

dbObj any object that implements some functionality in the R/S-Plus interface to databases (a driver, a connection or a result set).

res an PostgreSQLResult.

... currently not being used.

Note

nullOk in dbColumnInfo was changed. Now it may be TRUE, FALSE, or NA; the column may be totally deleted in future releases;

References

See the Database Interface definition document DBI.pdf in the base directory of this package or http://stat.bell-labs.com/RS-DBI.

See Also

PostgreSQL, dbDriver, dbConnect, dbSendQuery, dbGetQuery, fetch, dbCommit, dbGetInfo, dbListTables, dbReadTable.

Examples

```
## Not run:
drv <- dbDriver("PostgreSQL")
con <- dbConnect(drv, user= "user", password="password", dbname="sample")

dbListTables(con)

rs <- dbSendQuery(con, query.sql)
dbGetStatement(rs)
dbHasCompleted(rs)

info <- dbGetInfo(rs)
names(dbGetInfo(drv))

# DBIConnection info
names(dbGetInfo(con))

# DBIResult info
```

```
names(dbGetInfo(rs))

## End(Not run)
```

---

dbListTables-methods *List items from an PostgreSQL DBMS and from objects*

Description

These methods are straight-forward implementations of the corresponding generic functions.

Methods

drv an PostgreSQLDriver.

conn an PostgreSQLConnection.

name a character string with the table name.

... currently not used.

References

See the Database Interface definition document DBI.pdf in the base directory of this package or http://stat.bell-labs.com/RS-DBI.

See Also

PostgreSQL, dbGetInfo, dbColumnInfo, dbDriver, dbConnect, dbSendQuery

Examples

```
## Not run:
drv <- dbDriver("PostgreSQL")
# after working awhile...
for(con in dbListConnections(drv)){
   dbGetStatement(dbListResults(con))
}

## End(Not run)
```

dbObjectId-class *Class dbObjectId*

Description

A helper (mixin) class to provide external references in an R/S-Plus portable way.

Objects from the Class

A virtual Class: No objects may be created from it.

Slots

`Id`: Object of class `"integer"` this is an integer vector holding an opaque reference into a C struct (may or may not be a C pointer, may or may not have length one).

Methods

coerce `signature(from = "dbObjectId", to = "integer")`: ...

coerce `signature(from = "dbObjectId", to = "numeric")`: ...

coerce `signature(from = "dbObjectId", to = "character")`: ...

format `signature(x = "dbObjectId")`: ...

print `signature(x = "dbObjectId")`: ...

show `signature(object = "dbObjectId")`: ...

Note

A cleaner mechanism would use external references, but historically this class has existed mainly for R/S-Plus portability.

Examples

```
## Not run:
  pg <- dbDriver("PostgreSQL")
  con <- dbConnect(pg, "user", "password")
  is(pg, "dbObjectId")   ## True
  is(con, "dbObjectId")  ## True
  isPostgresqlIdCurrent(con)       ## True
  q("yes")
  $ R
  isPostgresqlIdCurrent(con)       ## False

## End(Not run)
```

dbReadTable-methods *Convenience functions for Importing/Exporting DBMS tables*

Description

These functions mimic their R/S-Plus counterpart `get`, `assign`, `exists`, `remove`, and `objects`, except that they generate code that gets remotely executed in a database engine.

Value

A `data.frame` in the case of `dbReadTable`; otherwise a logical indicating whether the operation was successful.

Methods

conn an `PostgreSQLConnection` database connection object.

name a character string specifying a table name.

value a data.frame (or coercible to data.frame).

When the `value` is a character string, it is assumed to be a file name containing the data to be loaded; The implementation is INCOMPLETE and should not be used in current state.

row.names UNTESTED;

in the case of `dbReadTable`, this argument can be a string or an index specifying the column in the DBMS table to be used as `row.names` in the output data.frame (a `NULL`, `""`, or 0 specifies that no column should be used as `row.names` in the output).

In the case of `dbWriteTable`, this argument should be a logical specifying whether the `row.names` should be output to the output DBMS table; if `TRUE`, an extra field whose name will be whatever the R/S-Plus identifier `"row.names"` maps to the DBMS (see `make.db.names`).

overwrite a logical specifying whether to overwrite an existing table or not. Its default is `FALSE`.

append a logical specifying whether to append to an existing table in the DBMS. Its default is `FALSE`.

allow.keywords `dbWriteTable` accepts a logical `allow.keywords` to allow or prevent PostgreSQL reserved identifiers to be used as column names. By default it is `FALSE`.

dots optional arguments.

When `dbWriteTable` is used to import data from a file, you may optionally specify `header=`, `row.names=`, `col.names=`, `sep=`, `eol=`, `field.types=`, `skip=`, and `quote=`. NOT FULLY IMPLEMENTED YET.

`header` is a logical indicating whether the first data line (but see `skip`) has a header or not. If missing, it value is determined following `read.table` convention, namely, it is set to TRUE if and only if the first row has one fewer field that the number of columns.

`row.names` is a logical to specify whether the first column is a set of row names. If missing its default follows the `read.table` convention.

`col.names` a character vector with column names; column names are quoted to work as SQL identifiers. Thus, the column names are case sensitive and `make.db.names` will NOT be used here.

sep= specifies the field separator, and its default is ','.

eol= specifies the end-of-line delimiter, and its default is '\n'.

skip specifies number of lines to skip before reading the data, and it defaults to 0.

field.types is a list of named field SQL types where names(field.types) provide the new table's column names (if missing, field types are inferred using dbDataType).

Note

dbWriteTable creates additional column in the database, while dbReadTable reads that column by default. So, it is not symmetrical in its current implementation. the backend raw_names may change in future versions.

References

See the Database Interface definition document DBI.pdf in the base directory of this package or http://stat.bell-labs.com/RS-DBI.

See Also

PostgreSQL, isSQLKeyword, dbDriver, dbConnect, dbSendQuery, dbGetQuery, fetch, dbCommit, dbGetInfo, dbListTables, dbReadTable.

Examples

```
## Not run:
conn <- dbConnect("PostgreSQL", dbname = "wireless")
if(dbExistsTable(con, "frame_fuel")){
   dbRemoveTable(conn, "frame_fuel")
   dbWriteTable(conn, "frame_fuel", fuel.frame)
}
if(dbExistsTable(conn, "RESULTS")){
   dbWriteTable(conn, "RESULTS", results2000, append = T)
else
   dbWriteTable(conn, "RESULTS", results2000)
}

## End(Not run)
```

---

dbSendQuery-methods *Execute a statement on a given database connection*

Description

These methods are straight-forward implementations of the corresponding generic functions. However, for complex data like array are just transferred as a string instead of the corresponding vector in R. This behavior will change in future releases, and the author is advised not to rely on it.

Methods

conn an PostgreSQLConnection object.

statement a character vector of length 1 with the SQL statement.

res an PostgreSQLResult object.

... additional parameters.

References

See the Database Interface definition document DBI.pdf in the base directory of this package or http://stat.bell-labs.com/RS-DBI.

See Also

PostgreSQL, dbDriver, dbConnect, fetch, dbCommit, dbGetInfo, dbReadTable.

Examples

```
## Not run:
drv <- dbDriver("PostgreSQL")
con <- dbConnect(drv, "usr", "password", "dbname")
res <- dbSendQuery(con, "SELECT * from sales")
data <- fetch(res, n = -1)
# alternatively, use dbGetQuery
data <- dbGetQuery(con, "SELECT * from sales")

## End(Not run)
```

---

dbSetDataMappings-methods

Set data mappings between PostgreSQL and R/S-Plus

Description

Not yet implemented

Methods

res a PostgreSQLResult object as returned by dbSendQuery.

flds a data.frame with field descriptions as returned by dbColumnInfo.

... any additional arguments are passed to the implementing method.

References

See the Database Interface definition document DBI.pdf in the base directory of this package or http://stat.bell-labs.com/RS-DBI.

See Also

PostgreSQL, dbSendQuery, fetch, dbColumnInfo.

Examples

```
## Not run:
makeImage <- function(x) {
  .C("make_Image", as.integer(x), length(x))
}

res <- dbSendQuery(con, statement)
flds <- dbColumnInfo(res)
flds[3, "Sclass"] <- makeImage

dbSetDataMappings(rs, flds)

im <- fetch(rs, n = -1)

## End(Not run)
```

---

fetch-methods *Fetch records from a previously executed query*

Description

This method is a straight-forward implementation of the corresponding generic function.

Details

The RPostgreSQL implementations retrieves only n records, and if n is missing it only returns up to fetch.default.rec as specified in the call to PostgreSQL (500 by default).

Methods

res an PostgreSQLResult object.

n maximum number of records to retrieve per fetch. Use n = -1 to retrieve all pending records; use a value of n = 0 for fetching the default number of rows fetch.default.rec defined in the PostgreSQL initialization invocation.

... currently not used.

References

See the Database Interface definition document DBI.pdf in the base directory of this package or http://stat.bell-labs.com/RS-DBI.

See Also

PostgreSQL, dbConnect, dbSendQuery, dbGetQuery, dbClearResult, dbCommit, dbGetInfo, dbReadTable.

Examples

```
## Not run:
drv <- dbDriver("PostgreSQL")
con <- dbConnect(drv, user = "ruser",password = "123456",dbname = "status")
res <- dbSendQuery(con, statement = paste(
                      "SELECT w.category, w.cost, p.type",
                      "FROM items w, sales P",
                      "WHERE w.category = p.type",
                      "ORDER BY w.cost"))
# we now fetch the first 100 records from the resultSet into a data.frame
data1 <- fetch(res, n = 100)
dim(data1)

dbHasCompleted(res)

# let's get all remaining records
data2 <- fetch(res, n = -1)

## End(Not run)
```

---

isPostgresqlIdCurrent *Check whether a database handle object is valid or not*

Description

Support function that verifies that an object holding a reference to a foreign object is still valid for communicating with the RDBMS

Usage

```
isPostgresqlIdCurrent(obj)
```

Arguments

obj	any dbObject (e.g., dbDriver, dbConnection, dbResult).

Details

dbObjects are R/S-Plus remote references to foreign objects. This introduces differences to the object's semantics such as persistence (e.g., connections may be closed unexpectedly), thus this function provides a minimal verification to ensure that the foreign object being referenced can be contacted.

Value

a logical scalar.

See Also

dbDriver dbConnect dbSendQuery fetch

Examples

```
## Not run:
cursor <- dbSendQuery(con, sql.statement)
isIdCurrent(cursor)

## End(Not run)
```

`make.db.names-methods` *Make R/S-Plus identifiers into quoted PostgreSQL identifiers*

Description

Calls postgresqlquoteId to make valid quoted identifiers. This has calling convention same as the make.db.names for compatibility.

Methods

dbObj any PostgreSQL object (e.g., `PostgreSQLDriver`). Just ignored.

snames a character vector of R/S-Plus identifiers (symbols) from which we need to make SQL identifiers.

name a character vector of SQL identifiers we want to check against keywords from the DBMS. Ignored.

unique logical describing whether the resulting set of SQL names should be unique. Its default is `TRUE`. Following the SQL 92 standard, uniqueness of SQL identifiers is determined regardless of whether letters are upper or lower case. Ignored.

allow.keywords logical describing whether SQL keywords should be allowed in the resulting set of SQL names. Its default is `TRUE`. Ignored.

keywords a character vector with SQL keywords, by default it is `.PostgreSQLKeywords` define in `RPostgreSQL`. This may be overriden by users. Ignored.

case a character string specifying whether to make the comparison as lower case, upper case, or any of the two. it defaults to `any`. Ignored.

... currently not used.

References

The set of SQL keywords is stored in the character vector `.SQL92Keywords` and reflects the SQL ANSI/ISO standard as documented in "X/Open SQL and RDA", 1994, ISBN 1-872630-68-8. Users can easily override or update this vector.

PostgreSQL does add some keywords to the SQL 92 standard, they are listed in the `.PostgreSQLKeywords` object.

See the Database Interface definition document `DBI.pdf` in the base directory of this package or http://stat.bell-labs.com/RS-DBI.

See Also

PostgreSQL, dbReadTable, dbWriteTable, dbExistsTable, dbRemoveTable, dbListTables.

Examples

```
## Not run:
# This example shows how we could export a bunch of data.frames
# into tables on a remote database.


## End(Not run)
```

PostgreSQL *Instantiate a PostgreSQL client from the current R or S-Plus session*

Description

This function creates and initializes a PostgreSQL client. It returns an driver object that allows you to connect to one or several PostgreSQL servers.

Usage

```
PostgreSQL(max.con = 16, fetch.default.rec = 500, force.reload = FALSE)
```

Arguments

max.con	Maximum number of connections that are intended to have open at one time. There's no intrinic limit, since strictly speaking this limit applies to PostgreSQL *servers*, but clients can have (at least in theory) more than this. Typically there are at most a handful of open connections, thus the internal RPostgreSQL code uses a very simple linear search algorithm to manage its connection table.
fetch.default.rec	number of records to fetch at one time from the database. (The fetch method uses this number as a default.)
force.reload	should the client code be reloaded (reinitialize)? Setting this to TRUE allows you to change default settings. Notice that all connections should be closed before re-loading.

Details

This object is a singleton, that is, on subsequent invocations it returns the same initialized object.

This implementation allows you to connect to multiple host servers and run multiple connections on each server simultaneously.

Value

An object PostgreSQLDriver that extends dbDriver and dbObjectId. This object is required to create connections to one or several PostgreSQL database engines.

Side Effects

The R/S-Plus client part of the database communication is initialized, but note that connecting to the database engine needs to be done through calls to dbConnect.

User authentication

The passed string can be empty to use all default parameters, or it can contain one or more parameter settings separated by comma. Each parameter setting is in the form parameter = "value". Spaces around the equal sign are optional.

The most important parameters are user, password, host, dbname, port, tty and options.

References

See stat.bell-labs.com/RS-DBI for more details on the R/S-Plus database interface.

See the documentation at the PostgreSQL Web site http://www.postgresql.org for details.

Author(s)

David A. James

See Also

On database managers:

dbDriver dbUnloadDriver

On connections, SQL statements and resultSets:

dbConnect dbDisconnect dbSendQuery dbGetQuery fetch dbClearResult

On transaction management:

dbCommit dbRollback

On meta-data:

summary dbGetInfo dbGetDBIVersion dbListTables dbListConnections dbListResults dbColumnInfo dbGetException dbGetStatement dbHasCompleted dbGetRowCount dbGetRowsAffected

Examples

```
## Not run:
# create a PostgreSQL instance and create one connection.
> m <- dbDriver("PostgreSQL")
<PostgreSQLDriver:(4378)>

> con <- dbConnect(m, user="username", password="passwd", dbname="database_name")
> rs <- dbSendQuery(con, "select * sales where price < 10")
> df <- fetch(rs, n = 50)
> dbHasCompleted(rs)
[1] FALSE
> df2 <- fetch(rs, n = -1)
> dbHasCompleted(rs)
[1] TRUE
```

```
> dbClearResult(rs)
> dbListTables(con)

## End(Not run)
```

`postgresqlBuildTableDefinition` *Build the SQL CREATE TABLE definition as a string*

Description

Build the SQL CREATE TABLE definition as a string for the input data.frame

Usage

```
postgresqlBuildTableDefinition(dbObj, name, obj, field.types = NULL, row.names = TRUE, ...)
```

Arguments

`dbObj`	any DBI object (used only to dispatch according to the engine (e.g., MySQL, Oracle, PostgreSQL, SQLite)
`name`	name of the new SQL table
`obj`	an R object coerceable to data.frame for which we want to create a table
`field.types`	optional named list of the types for each field in `obj`
`row.names`	logical, should row.name of `value` be exported as a `row_names` field? Default is TRUE
`...`	reserved for future use

Details

The output SQL statement is a simple `CREATE TABLE` with suitable for `dbGetQuery`

Value

An SQL string

References

See the Database Interface definition document `DBI.pdf` in the base directory of this package or http://stat.bell-labs.com/RS-DBI.

See Also

PostgreSQL, dbConnect, dbSendQuery, dbGetQuery, fetch, dbCommit, dbGetInfo, dbReadTable.

PostgreSQLConnection-class
Class PostgreSQLConnection

Description

PostgreSQLConnection class.

Generators

The method dbConnect is the main generator.

Extends

Class "DBIConnection", directly. Class "PostgreSQLObject", directly. Class "DBIObject", by class "DBIConnection". Class "dbObjectId", by class "PostgreSQLObject".

Methods

coerce signature(from = "PostgreSQLConnection", to = "PostgreSQLResult"): ...

dbCallProc signature(conn = "PostgreSQLConnection"): ...

dbCommit signature(conn = "PostgreSQLConnection"): ...

dbConnect signature(drv = "PostgreSQLConnection"): ...

dbDisconnect signature(conn = "PostgreSQLConnection"): ...

dbExistsTable signature(conn = "PostgreSQLConnection", name = "character"): ...

dbGetException signature(conn = "PostgreSQLConnection"): ...

dbGetInfo signature(dbObj = "PostgreSQLConnection"): ...

dbGetQuery signature(conn = "PostgreSQLConnection", statement = "character"): ...

dbListFields signature(conn = "PostgreSQLConnection", name = "character"): ...

dbListResults signature(conn = "PostgreSQLConnection"): ...

dbListTables signature(conn = "PostgreSQLConnection"): ...

dbReadTable signature(conn = "PostgreSQLConnection", name = "character"): ...

dbRemoveTable signature(conn = "PostgreSQLConnection", name = "character"): ...

dbRollback signature(conn = "PostgreSQLConnection"): ...

dbSendQuery signature(conn = "PostgreSQLConnection", statement = "character"): ...

dbWriteTable signature(conn = "PostgreSQLConnection", name = "character", value = "data.frame"): ...

summary signature(object = "PostgreSQLConnection"): ...

References

See the Database Interface definition document DBI.pdf in the base directory of this package or http://developer.r-project.org/db.

See Also

DBI base classes:

DBIObject-class DBIDriver-class DBIConnection-class DBIResult-class

PostgreSQL classes:

PostgreSQLObject-class PostgreSQLDriver-class PostgreSQLConnection-class PostgreSQLResult-class

Examples

```
## Not run:
drv <- dbDriver("PostgreSQL")
con <- dbConnect(drv, dbname = "template1")

## End(Not run)
```

---

postgresqlDBApply *Apply R/S-Plus functions to remote groups of DBMS rows (experimental)*

Description

Applies R/S-Plus functions to groups of remote DBMS rows without bringing an entire result set all at once. The result set is expected to be sorted by the grouping field.

Usage

```
postgresqlDBApply(res, INDEX, FUN = stop("must specify FUN"),
         begin = NULL,
         group.begin =  NULL,
         new.record = NULL,
         end = NULL,
         batchSize = 100, maxBatch = 1e6,
         ..., simplify = TRUE)
```

Arguments

res	a result set (see dbSendQuery).
INDEX	a character or integer specifying the field name or field number that defines the various groups.
FUN	a function to be invoked upon identifying the last row from every group. This function will be passed a data frame holding the records of the current group, a character string with the group label, plus any other arguments passed to dbApply as "...".

`begin`	a function of no arguments to be invoked just prior to retrieve the first row from the result set.
`end`	a function of no arguments to be invoked just after retrieving the last row from the result set.
`group.begin`	a function of one argument (the group label) to be invoked upon identifying a row from a new group.
`new.record`	a function to be invoked as each individual record is fetched. The first argument to this function is a one-row data.frame holding the new record.
`batchSize`	the default number of rows to bring from the remote result set. If needed, this is automatically extended to hold groups bigger than `batchSize`.
`maxBatch`	the absolute maximum of rows per group that may be extracted from the result set.
`...`	any additional arguments to be passed to `FUN`.
`simplify`	Not yet implemented

Details

`dbApply` This function is meant to handle somewhat gracefully(?) large amounts of data from the DBMS by bringing into R manageable chunks (about `batchSize` records at a time, but not more than `maxBatch`); the idea is that the data from individual groups can be handled by R, but not all the groups at the same time.

The PostgreSQL implementation `postgresqlDBApply` allows us to register R functions that get invoked when certain fetching events occur. These include the "begin" event (no records have been yet fetched), "begin.group" (the record just fetched belongs to a new group), "new record" (every fetched record generates this event), "group.end" (the record just fetched was the last row of the current group), "end" (the very last record from the result set). Awk and perl programmers will find this paradigm very familiar (although SAP's ABAP language is closer to what we're doing).

Value

A list with as many elements as there were groups in the result set.

Note

This is an experimental version implemented only in R (there are plans, time permitting, to implement it in S-Plus).

The terminology that we're using is closer to SQL than R. In R what we're referring to "groups" are the individual levels of a factor (grouping field in our terminology).

See Also

`PostgreSQL`, `dbSendQuery`, `fetch`.

Examples

```
## Not run:
drv <- dbDriver(RPostgreSQL)
con <- dbConnect(drv, user ="usrname", password="pword", dname="database")
res <- dbSendQuery(con,
             "select Agent, ip_addr, DATA from pseudo_data order by Agent")
out <- dbApply(res, INDEX = "Agent",
        FUN = function(x, grp) quantile(x$DATA, names=FALSE))

## End(Not run)
```

PostgreSQLDriver-class

Class PostgreSQLDriver

Description

An PostgreSQL driver implementing the R/S-Plus database (DBI) API.

Generators

The main generators are dbDriver and PostgreSQL.

Extends

Class "DBIDriver", directly. Class "PostgreSQLObject", directly. Class "DBIObject", by class "DBIDriver". Class "dbObjectId", by class "PostgreSQLObject".

Methods

coerce signature(from = "PostgreSQLObject", to = "PostgreSQLDriver"): ...

dbConnect signature(drv = "PostgreSQLDriver"): ...

dbGetInfo signature(dbObj = "PostgreSQLDriver"): ...

dbListConnections signature(drv = "PostgreSQLDriver"): ...

dbUnloadDriver signature(drv = "PostgreSQLDriver"): ...

summary signature(object = "PostgreSQLDriver"): ...

References

See the Database Interface definition document DBI.pdf in the base directory of this package or http://developer.r-project.org/db.

See Also

DBI base classes:

DBIObject-class DBIDriver-class DBIConnection-class DBIResult-class

PostgreSQL classes:

PostgreSQLObject-class PostgreSQLDriver-class PostgreSQLConnection-class PostgreSQLResult-class

Examples

```
## Not run:
drv <- dbDriver("PostgreSQL")
con <- dbConnect(drv, dbname="template1")

## End(Not run)
```

PostgreSQLObject-class

Class PostgreSQLObject

Description

Base class for all PostgreSQL-specific DBI classes

Objects from the Class

A virtual Class: No objects may be created from it.

Extends

Class "DBIObject", directly. Class "dbObjectId", directly.

Methods

coerce signature(from = "PostgreSQLObject", to = "PostgreSQLriver"): ...

dbDataType signature(dbObj = "PostgreSQLObject"): ...

isSQLKeyword signature(dbObj = "PostgreSQLObject", name = "character"): ...

make.db.names signature(dbObj = "PostgreSQLObject", snames = "character"): ...

SQLKeywords signature(dbObj = "PostgreSQLObject"): ...

References

See the Database Interface definition document DBI.pdf in the base directory of this package or http://developer.r-project.org/db.

See Also

DBI base classes:

DBIObject-class DBIDriver-class DBIConnection-class DBIResult-class

PostgreSQL classes:

PostgreSQLObject-class PostgreSQLDriver-class PostgreSQLConnection-class PostgreSQLResult-class

Examples

```
## Not run:
drv <- dbDriver("PostgreSQL")
con <- dbConnect(drv, dbname = "template1")

## End(Not run)
```

PostgreSQLResult-class

Class PostgreSQLResult

Description

PostgreSQL's query results class. This classes encapsulates the result of an SQL statement (either `select` or not).

Generators

The main generator is `dbSendQuery`.

Extends

Class `"DBIResult"`, directly. Class `"PostgreSQLObject"`, directly. Class `"DBIObject"`, by class "DBIResult". Class `"dbObjectId"`, by class "PostgreSQLObject".

Methods

coerce `signature(from = "PostgreSQLConnection", to = "PostgreSQLResult")`: ...

dbClearResult `signature(res = "PostgreSQLResult")`: ...

dbColumnInfo `signature(res = "PostgreSQLResult")`: ...

dbGetException `signature(conn = "PostgreSQLResult")`: ...

dbGetInfo `signature(dbObj = "PostgreSQLResult")`: ...

dbGetRowCount `signature(res = "PostgreSQLResult")`: ...

dbGetRowsAffected `signature(res = "PostgreSQLResult")`: ...

dbGetStatement `signature(res = "PostgreSQLResult")`: ...

dbHasCompleted `signature(res = "PostgreSQLResult")`: ...

dbListFields `signature(conn = "PostgreSQLResult", name = "missing")`: ...

fetch signature(res = "PostgreSQLResult", n = "numeric"): ...

fetch signature(res = "PostgreSQLResult", n = "missing"): ...

summary signature(object = "PostgreSQLResult"): ...

References

See the Database Interface definition document DBI.pdf in the base directory of this package or http://developer.r-project.org/db

See Also

DBI base classes:

DBIObject-class DBIDriver-class DBIConnection-class DBIResult-class

PostgreSQL classes:

PostgreSQLObject-class PostgreSQLDriver-class PostgreSQLConnection-class PostgreSQLResult-class

Examples

```
## Not run:
drv <- dbDriver("PostgreSQL")
con <- dbConnect(drv, dbname = "template1")
## rs is the result set
rs <- dbSendQuery(con,"select * from sales")
## display the first three values of result set
fetch(rs,n=3)

## End(Not run)
```

---

postgresqlSupport *Support Functions*

Description

These functions are the workhorse behind the RPostgreSQL package, but users need not invoke these directly. For details see PostgreSQL.

Usage

```
## PostgreSQLDriver-related
postgresqlInitDriver(max.con=16, fetch.default.rec = 500, force.reload=FALSE)
postgresqlDriverInfo(obj, what, ...)
postgresqlDescribeDriver(obj, verbose = FALSE, ...)
postgresqlCloseDriver(drv, ...)

## PostgreSQLConnection-related
postgresqlNewConnection(drv, user, password, host, dbname, port, tty, options, forceISOdate = TRUE)
postgresqlCloneConnection(con, ...)
```

```
postgresqlConnectionInfo(obj, what, ...)
postgresqlDescribeConnection(obj, verbose = FALSE, ...)
postgresqlCloseConnection(con, ...)

## PostgreSQLResult-related
postgresqlExecStatement(con, statement, params, ...)
postgresqlFetch(res, n=0, ...)
postgresqlQuickSQL(con, statement, ...)
postgresqlResultInfo(obj, what, ...)
postgresqlDescribeResult(obj, verbose = FALSE, ...)
postgresqlCloseResult(res, ...)
postgresqlDescribeFields(res, ...)

## data mappings, convenience functions, and extensions
postgresqlDataType(obj, ...)
postgresqlReadTable(con, name, row.names = "row.names", check.names = TRUE, ...)
postgresqlWriteTable(con, name, value, field.types, row.names = TRUE,
   overwrite=FALSE, append=FALSE, ..., allow.keywords = FALSE)
postgresqlpqExec(con, statement)
postgresqlCopyIn(con, filename)
postgresqlgetResult(con)
postgresqlEscapeStrings(con, preescapedstring)
postgresqlQuoteId(identifiers)
postgresqlTableRef(identifiers)
postgresqlImportFile(con, name, value, field.types, overwrite=FALSE,
   append=FALSE, header, row.names, nrows=50, sep=",", eol="\n",
   skip = 0, quote="\"", ...)

## Transaction Management
postgresqlTransactionStatement(con, statement)
```

Arguments

max.con	positive integer specifying maximum number of open connections. The current default of 10 is hardcoded in the C code.
fetch.default.rec	default number of rows to fetch (move to R/S-Plus). This default is used in postgresqlFetch. The default is 500.
force.reload	logical indicating whether to re-initialize the driver. This may be useful if you want to change the defaults (e.g., fetch.default.rec). Note that the driver is a singleton (subsequent inits just returned the previously initialized driver, thus this argument).
obj	any of the PostgreSQL DBI objects (e.g., PostgreSQLConnection, PostgreSQLResult).
what	character vector of metadata to extract, e.g., "version", "statement", "isSelect".
verbose	logical controlling how much information to display. Defaults to FALSE.
drv	an PostgreSQLDriver object as produced by postgresqlInitDriver.
con	an PostgreSQLConnection object as produced by postgresqlNewConnection and postgresqlCloneConnection.

res	an PostgreSQLResult object as produced by by postgresqlExecStatement and postgresqlgetResult.
user	a character string with the PostgreSQL's user name.
password	character string with the PostgreSQL's password.
dbname	character string with the PostgreSQL database name.
host	character string with the name (or IP address) of the machine hosting the database. Default is "", which is interpreted as localhost by the PostgreSQL's API.
port	(optional) positive integer specifying the TCP port number that the PostgreSQL server is listening to. Consult the PostgreSQL documentation for details.
tty	Ignored (formerly, this specified where to send server debug output)
options	Command-line options to be sent to the server
forceISOdate	logical indicating whether "set datestyle to ISO" is issued upon connection. Although this is made as an option, the conversion needs to be ISO style for proper conversion of the date datatype.
force	logical indicating whether to close a connection that has open result sets. The default is FALSE.
statement	character string holding one (and only one) SQL statement.
params	actual values that is written as parameters in the statement.
n	number of rows to fetch from the given result set. A value of -1 indicates to retrieve all the rows. The default of 0 specifies to extract whatever the fetch.default.rec was specified during driver initialization postgresqlInit.
name	character vector of names (table names, fields, keywords).
value	a data.frame.
field.types	a list specifying the mapping from R/S-Plus fields in the data.frame value to SQL data types. The default is sapply(value, SQLDataType), see PostgreSQLSQLType.
header	logical, does the input file have a header line? Default is the same heuristic used by read.table, i.e., TRUE if the first line has one fewer column that the second line.
row.names	a logical specifying whether to prepend the value data.frame row names or not. The default is TRUE.
check.names	a logical specifying whether to convert DBMS field names into legal S names. Default is TRUE.
overwrite	logical indicating whether to replace the table name with the contents of the data.frame value. The defauls is FALSE.
append	logical indicating whether to append value to the existing table name.
nrows	number of lines to rows to import using read.table from the input file to create the proper table definition. Default is 50.
sep	field separator character.
eol	end-of-line separator.
skip	number of lines to skip before reading data in the input file.
quote	the quote character used in the input file (defaults to \".

allow.keywords	logical indicating whether column names that happen to be PostgreSQL keywords be used as column names in the resulting relation (table) being written. Defaults to FALSE, forcing postgresqlWriteTable to modify column names to make them legal PostgreSQL identifiers.
preescapedstring	character string to be escaped
identifiers	one or more character strings to be used as identfier in SQL statement
filename	character string indicating the file which contains the data to be copied to the PostgreSQL backend
...	placeholder for future use.

Value

postgresqlInitDriver returns an PostgreSQLDriver object.

postgresqlDriverInfo returns a list of name-value metadata pairs.

postgresqlDescribeDriver returns NULL (displays the object's metadata).

postgresqlCloseDriver returns a logical indicating whether the operation succeeded or not.

postgresqlNewConnection returns an PostgreSQLConnection object.

postgresqlCloneConnection returns an PostgreSQLConnection object.

postgresqlConnectionInforeturns a list of name-value metadata pairs.

postgresqlDescribeConnection returns NULL (displays the object's metadata).

postgresqlCloseConnection returns a logical indicating whether the operation succeeded or not.

postgresqlExecStatement returns an PostgreSQLResult object.

postgresqlFetch returns a data.frame.

postgresqlQuickSQL returns either a data.frame if the statement is a select-like or NULL otherwise.

postgresqlDescribeResult returns NULL (displays the object's metadata).

postgresqlCloseResult returns a logical indicating whether the operation succeeded or not.

postgresqlDescribeFields returns a data.frame with one row per field with columns name, Sclass, type, len, precision, scale, and nullOK which fully describe each field in a result set. Except for Sclass (which shows the mapping of the field type into an R/S-Plus class) all the information pertains to PostgreSQL's data storage attributes.

postgresqlReadTable returns a data.frame with the contents of the DBMS table.

postgresqlWriteTable returns a logical indicating whether the operation succeeded or not.

postgresqlpqExec returns NUL (executes the statement but does not try to get result. This is called internally from postgresqlWriteTable before postgresqlCopyInDataframe

postgresqlCopyIn returns NULL (copies the content of the file through the socket connection to postgresql backend. This should be used just after COPY tablename FROM STDIN query. This is not used now.)

postgresqlCopyInDataframe returns NULL (copies the content of the dataframe through the socket connection to postgresql backend. Strings are encoded as UTF-8 for transfer. The client_encoding should be set to UTF-8. This should be used just after COPY tablename FROM STDIN query.)

postgresqlgetResult returns an PostgreSQLResult object. This is called after completion of execution of postgresqlpqExec.

postgresqlEscapeStrings returns a character string which is escaped properly so that it can be surrounded with a single quote and used as literal in SQL. The escape procedure is dependent on the character encoding of the connection.

postgresqlQuoteId returns a character string which is quoted as identifier. Returns vector on vector arguemnt.

postgresqlTableRef returns a character string which is quoted as identifier. Reterns a charcter string concatenated with "." so that "dbname"."schemaname"."tablename" reference is created upon c("dbname", "schemaname", "tablename") arguemnt.

postgresqlDataType retuns a character string with the closest

postgresqlResultInfo returns a list of name-value metadata pairs.

postgresqlTransactionStatement returns a logical indicating whether the operation succeeded or not.

Constants

.PostgreSQLPkgName (currently "RPostgreSQL"), .PostgreSQLPkgVersion (the R package version), .PostgreSQLPkgRCS (the RCS revision), .PostgreSQL.NA.string (character that PostgreSQL uses to denote NULL on input), .PostgreSQLSQLKeywords (a lot!) .conflicts.OK.

---

summary-methods *Summarize an PostgreSQL object*

Description

These methods are straight-forward implementations of the corresponding generic functions.

Methods

object = "DBIObject" Provides relevant metadata information on object, for instance, the PostgreSQL server file, the SQL statement associated with a result set, etc.

from object to be coerced

to coercion class

x object to format or print or show

Index