Package 'RPostgres'

December 5, 2021

Title Rcpp Interface to PostgreSQL

Version 1.4.2

Date 2021-12-05

Description Fully DBI-compliant Rcpp-backed interface to PostgreSQL <https://www.postgresql.org/>, an open-source relational database.

License GPL-3

URL https://rpostgres.r-dbi.org, https://github.com/r-dbi/RPostgres

BugReports https://github.com/r-dbi/RPostgres/issues

Depends R (>= 3.1.0)

Imports bit64, blob (>= 1.2.0), DBI (>= 1.1.0), hms (>= 1.0.0), lubridate, methods, Rcpp (>= 1.0.7), withr

Suggests callr, covr, DBItest (>= 1.7.1), knitr, rmarkdown, testthat

LinkingTo plogr (>= 0.2.0), Rcpp

VignetteBuilder knitr

Encoding UTF-8

LazyLoad true

RoxygenNote 7.1.2

SystemRequirements libpq >= 9.0: libpq-dev (deb) or postgresql-devel (rpm)

Collate 'PqDriver.R' 'PqConnection.R' 'PqResult.R' 'RPostgres-pkg.R' 'RcppExports.R' 'Redshift.R' 'default.R' 'export.R' 'names.R' 'quote.R' 'tables.R' 'transactions.R' 'utils.R'

NeedsCompilation yes

Author Hadley Wickham [aut],
Jeroen Ooms [aut],
Kirill Müller [aut, cre] (<https://orcid.org/0000-0002-1416-3412>),
RStudio [cph],
R Consortium [fnd],
Tomoaki Nishiyama [ctb] (Code for encoding vectors into strings derived from RPostgreSQL)

Maintainer Kirill Müller <krlmlr+r@mailbox.org>
Repository CRAN
Date/Publication 2021-12-05 12:10:02 UTC

R topics documented:

`RPostgres-package` *RPostgres: Rcpp Interface to PostgreSQL*

Description

Fully DBI-compliant Rcpp-backed interface to PostgreSQL <https://www.postgresql.org/>, an open-source relational database.

Author(s)

Maintainer: Kirill Müller <krlmlr+r@mailbox.org> (ORCID)

Authors:

- Hadley Wickham
- Jeroen Ooms

Other contributors:

- RStudio [copyright holder]
- R Consortium [funder]
- Tomoaki Nishiyama (Code for encoding vectors into strings derived from RPostgreSQL) [contributor]

See Also

Useful links:

- https://rpostgres.r-dbi.org
- https://github.com/r-dbi/RPostgres
- Report bugs at https://github.com/r-dbi/RPostgres/issues

Postgres *Postgres driver*

Description

DBI::dbConnect() establishes a connection to a database. Set drv = RPostgres::Postgres() to connect to a SQL database using the **RPostgres** package.

Manually disconnecting a connection is not necessary with **RPostgres**, but still recommended; if you delete the object containing the connection, it will be automatically disconnected during the next GC with a warning.

Usage

```
Postgres()

## S4 method for signature 'PqDriver'
dbConnect(
  drv,
  dbname = NULL,
  host = NULL,
  port = NULL,
  password = NULL,
  user = NULL,
  service = NULL,
  ...,
  bigint = c("integer64", "integer", "numeric", "character"),
  check_interrupts = FALSE,
  timezone = "UTC",
  timezone_out = NULL
)

## S4 method for signature 'PqConnection'
dbDisconnect(conn, ...)
```

Arguments

drv	Should be set to Postgres() to use the **RPostgres** package.
dbname	Database name. If NULL, defaults to the user name. Note that this argument can only contain the database name, it will not be parsed as a connection string (internally, expand_dbname is set to false in the call to PQconnectdbParams()).
host, port	Host and port. If NULL, will be retrieved from PGHOST and PGPORT env vars.
user, password	User name and password. If NULL, will be retrieved from PGUSER and PGPASSWORD envvars, or from the appropriate line in ~/.pgpass. See https://www.postgresql.org/docs/current/libpq-pgpass.html for more details.

service — Name of service to connect as. If NULL, will be ignored. Otherwise, connection parameters will be loaded from the pg_service.conf file and used. See https://www.postgresql.org/docs/current/libpq-pgservice.html for details on this file and syntax.

... — Other name-value pairs that describe additional connection options as described at https://www.postgresql.org/docs/current/libpq-connect.html#LIBPQ-PARAMKEYWORDS

bigint — The R type that 64-bit integer types should be mapped to, default is bit64::integer64, which allows the full range of 64 bit integers.

check_interrupts — Should user interrupts be checked during the query execution (before first row of data is available)? Setting to TRUE allows interruption of queries running too long.

timezone — Sets the timezone for the connection. The default is "UTC". If NULL then no timezone is set, which defaults to the server's time zone.

timezone_out — The time zone returned to R, defaults to timezone. If you want to display datetime values in the local timezone, set to Sys.timezone() or "". This setting does not change the time values returned, only their display.

conn — Connection to disconnect.

Examples

```
library(DBI)
# Pass more arguments as necessary to dbConnect()
con <- dbConnect(RPostgres::Postgres())
dbDisconnect(con)
```

---

postgres-query *Execute a SQL statement on a database connection*

Description

To retrieve results a chunk at a time, use dbSendQuery(), dbFetch(), then dbClearResult(). Alternatively, if you want all the results (and they'll fit in memory) use dbGetQuery() which sends, fetches and clears for you.

Usage

```
## S4 method for signature 'PqConnection'
dbSendQuery(conn, statement, params = NULL, ..., immediate = FALSE)

## S4 method for signature 'PqResult'
dbFetch(res, n = -1, ..., row.names = FALSE)

## S4 method for signature 'PqResult'
```

```
dbBind(res, params, ...)

## S4 method for signature 'PqResult'
dbHasCompleted(res, ...)

## S4 method for signature 'PqResult'
dbClearResult(res, ...)
```

Arguments

conn	A PqConnection created by dbConnect().
statement	An SQL string to execute.
params	A list of query parameters to be substituted into a parameterised query. Query parameters are sent as strings, and the correct type is imputed by PostgreSQL. If this fails, you can manually cast the parameter with e.g. "$1::bigint".
...	Other arguments needed for compatibility with generic (currently ignored).
immediate	If TRUE, uses the PGsendQuery() API instead of PGprepare(). This allows to pass multiple statements and turns off the ability to pass parameters.
res	Code a PqResult produced by DBI::dbSendQuery().
n	Number of rows to return. If less than zero returns all rows.
row.names	Either TRUE, FALSE, NA or a string. If TRUE, always translate row names to a column called "row_names". If FALSE, never translate row names. If NA, translate rownames only if they're a character vector. A string is equivalent to TRUE, but allows you to override the default name. For backward compatibility, NULL is equivalent to FALSE.

Multiple queries and statements

With immediate = TRUE, it is possible to pass multiple queries or statements, separated by semicolons. For multiple statements, the resulting value of dbGetRowsAffected() corresponds to the total number of affected rows. If multiple queries are used, all queries must return data with the same column names and types. Queries and statements can be mixed.

Examples

```
library(DBI)
db <- dbConnect(RPostgres::Postgres())
dbWriteTable(db, "usarrests", datasets::USArrests, temporary = TRUE)

# Run query to get results as dataframe
dbGetQuery(db, "SELECT * FROM usarrests LIMIT 3")

# Send query to pull requests in batches
res <- dbSendQuery(db, "SELECT * FROM usarrests")
dbFetch(res, n = 2)
dbFetch(res, n = 2)
```

```
dbHasCompleted(res)
dbClearResult(res)

dbRemoveTable(db, "usarrests")

dbDisconnect(db)
```

---

postgres-tables *Convenience functions for reading/writing DBMS tables*

Description

dbWriteTable() executes several SQL statements that create/overwrite a table and fill it with values. **RPostgres** does not use parameterised queries to insert rows because benchmarks revealed that this was considerably slower than using a single SQL string.

dbAppendTable() is overridden because **RPostgres** uses placeholders of the form $1, $2 etc. instead of ?.

Usage

```
## S4 method for signature 'PqConnection,character,data.frame'
dbWriteTable(
  conn,
  name,
  value,
  ...,
  row.names = FALSE,
  overwrite = FALSE,
  append = FALSE,
  field.types = NULL,
  temporary = FALSE,
  copy = NULL
)

## S4 method for signature 'PqConnection'
sqlData(con, value, row.names = FALSE, ...)

## S4 method for signature 'PqConnection'
dbAppendTable(conn, name, value, copy = NULL, ..., row.names = NULL)

## S4 method for signature 'PqConnection,character'
dbReadTable(conn, name, ..., check.names = TRUE, row.names = FALSE)

## S4 method for signature 'PqConnection'
dbListTables(conn, ...)
```

```
## S4 method for signature 'PqConnection,character'
dbExistsTable(conn, name, ...)

## S4 method for signature 'PqConnection,Id'
dbExistsTable(conn, name, ...)

## S4 method for signature 'PqConnection,character'
dbRemoveTable(conn, name, ..., temporary = FALSE, fail_if_missing = TRUE)

## S4 method for signature 'PqConnection,character'
dbListFields(conn, name, ...)

## S4 method for signature 'PqConnection,Id'
dbListFields(conn, name, ...)

## S4 method for signature 'PqConnection'
dbListObjects(conn, prefix = NULL, ...)
```

Arguments

conn	a PqConnection object, produced by DBI::dbConnect()
name	a character string specifying a table name. Names will be automatically quoted so you can use any sequence of characters, not just any valid bare table name. Alternatively, pass a name quoted with dbQuoteIdentifier(), an Id() object, or a string escaped with SQL().
value	A data.frame to write to the database.
...	Ignored.
row.names	Either TRUE, FALSE, NA or a string. If TRUE, always translate row names to a column called "row_names". If FALSE, never translate row names. If NA, translate rownames only if they're a character vector. A string is equivalent to TRUE, but allows you to override the default name. For backward compatibility, NULL is equivalent to FALSE.
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.
field.types	character vector of named SQL field types where the names are the names of new table's columns. If missing, types are inferred with DBI::dbDataType()). The types can only be specified with append = FALSE.
temporary	If TRUE, only temporary tables are considered.
copy	If TRUE, serializes the data frame to a single string and uses COPY name FROM stdin. This is fast, but not supported by all postgres servers (e.g. Amazon's Redshift). If FALSE, generates a single SQL string. This is slower, but always supported. The default maps to TRUE on connections established via Postgres() and to FALSE on connections established via Redshift().

con	A database connection.
check.names	If TRUE, the default, column names will be converted to valid R identifiers.
fail_if_missing	If FALSE, dbRemoveTable() succeeds if the table doesn't exist.
prefix	A fully qualified path in the database's namespace, or NULL. This argument will be processed with dbUnquoteIdentifier(). If given the method will return all objects accessible through this prefix.

Schemas, catalogs, tablespaces

Pass an identifier created with Id() as the name argument to specify the schema or catalog, e.g. name = Id(catalog = "my_catalog",schema = "my_schema",table = "my_table") . To specify the tablespace, use dbExecute(conn,"SET default_tablespace TO my_tablespace") before creating the table.

Examples

```
library(DBI)
con <- dbConnect(RPostgres::Postgres())
dbListTables(con)
dbWriteTable(con, "mtcars", mtcars, temporary = TRUE)
dbReadTable(con, "mtcars")

dbListTables(con)
dbExistsTable(con, "mtcars")

# A zero row data frame just creates a table definition.
dbWriteTable(con, "mtcars2", mtcars[0, ], temporary = TRUE)
dbReadTable(con, "mtcars2")

dbDisconnect(con)
```

---

postgres-transactions *Transaction management.*

Description

dbBegin() starts a transaction. dbCommit() and dbRollback() end the transaction by either committing or rolling back the changes.

Usage

```
## S4 method for signature 'PqConnection'
dbBegin(conn, ..., name = NULL)

## S4 method for signature 'PqConnection'
```

```
dbCommit(conn, ..., name = NULL)

## S4 method for signature 'PqConnection'
dbRollback(conn, ..., name = NULL)
```

Arguments

conn	a PqConnection object, produced by DBI::dbConnect()
...	Unused, for extensibility.
name	If provided, uses the SAVEPOINT SQL syntax to establish, remove (commit) or undo a ßsavepoint.

Value

A boolean, indicating success or failure.

Examples

```
library(DBI)
con <- dbConnect(RPostgres::Postgres())
dbWriteTable(con, "USarrests", datasets::USArrests, temporary = TRUE)
dbGetQuery(con, 'SELECT count(*) from "USarrests"')

dbBegin(con)
dbExecute(con, 'DELETE from "USarrests" WHERE "Murder" > 1')
dbGetQuery(con, 'SELECT count(*) from "USarrests"')
dbRollback(con)

# Rolling back changes leads to original count
dbGetQuery(con, 'SELECT count(*) from "USarrests"')

dbRemoveTable(con, "USarrests")
dbDisconnect(con)
```

---

postgresHasDefault *Check if default database is available.*

Description

RPostgres examples and tests connect to a default database via dbConnect(Postgres()). This function checks if that database is available, and if not, displays an informative message.

postgresDefault() works similarly but returns a connection on success and throws a testthat skip condition on failure, making it suitable for use in tests.

Usage

```
postgresHasDefault(...)

postgresDefault(...)
```

Arguments

...	Additional arguments passed on to dbConnect()

Examples

```
if (postgresHasDefault()) {
  db <- postgresDefault()
  print(dbListTables(db))
  dbDisconnect(db)
} else {
  message("No database connection.")
}
```

---

postgresWaitForNotify *Wait for and return any notifications that return within timeout*

Description

Once you subscribe to notifications with LISTEN, use this to wait for responses on each channel.

Usage

```
postgresWaitForNotify(conn, timeout = 1)
```

Arguments

conn	a PqConnection object, produced by DBI::dbConnect()
timeout	How long to wait, in seconds. Default 1

Value

If a notification was available, a list of:

channel Name of channel

pid PID of notifying server process

payload Content of notification

If no notifications are available, return NULL

Examples

```
library(DBI)
library(callr)

# listen for messages on the grapevine
db_listen <- dbConnect(RPostgres::Postgres())
dbExecute(db_listen, "LISTEN grapevine")

# Start another process, which sends a message after a delay
rp <- r_bg(function () {
    library(DBI)
    Sys.sleep(0.3)
    db_notify <- dbConnect(RPostgres::Postgres())
    dbExecute(db_notify, "NOTIFY grapevine, 'psst'")
    dbDisconnect(db_notify)
})

# Sleep until we get the message
n <- NULL
while (is.null(n)) {
    n <- RPostgres::postgresWaitForNotify(db_listen, 60)
}
stopifnot(n$payload == 'psst')

# Tidy up
rp$wait()
dbDisconnect(db_listen)
```

---

quote *Quote postgres strings, identifiers, and literals*

Description

If an object of class Id is used for `dbQuoteIdentifier()`, it needs at most one `table` component and at most one `schema` component.

Usage

```
## S4 method for signature 'PqConnection,character'
dbQuoteString(conn, x, ...)

## S4 method for signature 'PqConnection,SQL'
dbQuoteString(conn, x, ...)

## S4 method for signature 'PqConnection,character'
dbQuoteIdentifier(conn, x, ...)
```

```
## S4 method for signature 'PqConnection,SQL'
dbQuoteIdentifier(conn, x, ...)

## S4 method for signature 'PqConnection,Id'
dbQuoteIdentifier(conn, x, ...)

## S4 method for signature 'PqConnection,SQL'
dbUnquoteIdentifier(conn, x, ...)

## S4 method for signature 'PqConnection'
dbQuoteLiteral(conn, x, ...)
```

Arguments

conn	A PqConnection created by dbConnect()
x	A character vector to be quoted.
...	Other arguments needed for compatibility with generic (currently ignored).

Examples

```
library(DBI)
con <- dbConnect(RPostgres::Postgres())

x <- c("a", "b c", "d'e", "\\f")
dbQuoteString(con, x)
dbQuoteIdentifier(con, x)
dbDisconnect(con)
```

Redshift *Redshift driver/connection*

Description

Use Redshift() instead of Postgres() to connect to an AWS Redshift cluster. All methods in **RPostgres** and downstream packages can be called on such connections. Some have different behavior for Redshift connections, to ensure better interoperability.

Usage

```
Redshift()

## S4 method for signature 'RedshiftDriver'
dbConnect(
  drv,
  dbname = NULL,
```

```
  host = NULL,
  port = NULL,
  password = NULL,
  user = NULL,
  service = NULL,
  ...,
  bigint = c("integer64", "integer", "numeric", "character"),
  check_interrupts = FALSE,
  timezone = "UTC"
)
```

Arguments

drv	Should be set to Postgres() to use the **RPostgres** package.
dbname	Database name. If NULL, defaults to the user name. Note that this argument can only contain the database name, it will not be parsed as a connection string (internally, expand_dbname is set to false in the call to PQconnectdbParams()).
host	Host and port. If NULL, will be retrieved from PGHOST and PGPORT env vars.
port	Host and port. If NULL, will be retrieved from PGHOST and PGPORT env vars.
password	User name and password. If NULL, will be retrieved from PGUSER and PGPASSWORD envvars, or from the appropriate line in ~/.pgpass. See https://www.postgresql.org/docs/current/libpq-pgpass.html for more details.
user	User name and password. If NULL, will be retrieved from PGUSER and PGPASSWORD envvars, or from the appropriate line in ~/.pgpass. See https://www.postgresql.org/docs/current/libpq-pgpass.html for more details.
service	Name of service to connect as. If NULL, will be ignored. Otherwise, connection parameters will be loaded from the pg_service.conf file and used. See https://www.postgresql.org/docs/current/libpq-pgservice.html for details on this file and syntax.
...	Other name-value pairs that describe additional connection options as described at https://www.postgresql.org/docs/current/libpq-connect.html#LIBPQ-PARAMKEYWORDS
bigint	The R type that 64-bit integer types should be mapped to, default is bit64::integer64, which allows the full range of 64 bit integers.
check_interrupts	Should user interrupts be checked during the query execution (before first row of data is available)? Setting to TRUE allows interruption of queries running too long.
timezone	Sets the timezone for the connection. The default is "UTC". If NULL then no timezone is set, which defaults to the server's time zone.

Index