Package ‘RPostgres’

October 22, 2023

Title   C++ Interface to PostgreSQL
Version  1.4.6
Date    2023-10-10
Description Fully DBI-compliant Rcpp-backed interface to PostgreSQL
<https://www.postgresql.org/>, an open-source relational database.
License   MIT + file LICENSE
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, withr
Suggests callr, covr, DBItest (>= 1.7.2.9001), knitr, rmarkdown,
testthat (>= 3.0.0)
LinkingTo  cpp11, plogr (>= 0.2.0)
VignetteBuilder knitr
Config/autostyle/scope line_breaks
Config/autostyle/strict  false
Config/testthat/edition  3
Encoding UTF-8
LazyLoad  true
RoxygenNote  7.2.3
SystemRequirements  libpq >= 9.0: libpq-dev (deb) or postgresql-devel (rpm)
   ‘Redshift.R’ ‘cpp11.R’ ‘dbAppendTable_PqConnection.R’
   ‘dbBegin_PqConnection.R’ ‘dbBind_PqResult.R’
   ‘dbClearResult_PqResult.R’ ‘dbColumnInfo_PqResult.R’
   ‘dbCommit_PqConnection.R’ ‘dbConnect_PqDriver.R’
   ‘dbConnect_RedshiftDriver.R’ ‘dbDataType_PqConnection.R’
R topics documented:

'RPostgres-package' .................................................. 3
Postgres ................................................................. 4
postgres-query .......................................................... 5
postgres-tables ......................................................... 7
postgres-transactions ................................................. 9
postgresHasDefault .................................................... 10
postgresIsTransacting ................................................ 11
Description

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

Author(s)

Maintainer: Kirill Müller <kirill@cynkra.com> (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://rpostres.r-dbi.org
- https://github.com/r-dbi/RPostgres
Postgres driver

Description

DBI::dbConnect() establishes a connection to a database. Set \( \text{drv} = \text{Postgres()} \) to connect to a PostgreSQL(-ish) database. Use \( \text{drv} = \text{Redshift()} \) instead to connect to an AWS Redshift cluster. Manually disconnecting a connection is not necessary with \text{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()

```r
## S4 method for signature 'PqDriver'

dbConnect(
  drv,
  dbname = NULL,
  host = NULL,
  port = NULL,
  password = NULL,
  user = NULL,
  service = NULL,
  ...
)

## S4 method for signature 'PqConnection'

dbDisconnect(conn, ...)
```

Arguments

- \( \text{drv} \): DBI::DBIDriver. Use \text{Postgres()} to connect to a PostgreSQL(-ish) database or \text{Redshift()} to connect to an AWS Redshift cluster. Use an existing DBI::DBIConnection object to clone an existing connection.
- \( \text{dbname} \): 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()).
- \( \text{host, port} \): Host and port. If NULL, will be retrieved from PGHOST and PGPORT env vars.
- \( \text{user, password} \): User name and password. If NULL, will be retrieved from PGUSER and PGPASSWORD env vars, or from the appropriate line in \~/.pgpass. See [https://www.postgresql.org/docs/current/libpq-pgpass.html](https://www.postgresql.org/docs/current/libpq-pgpass.html) for more details.
postgres-query

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

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.

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.

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.

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

Connection to disconnect.

Examples

```r
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

```r
## S4 method for signature 'PqResult'
dbBind(res, params, ...)

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

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

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

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

### Arguments

- **res**: Code a PqResult produced by DBI::dbSendQuery().
- **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).
- **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.
- **conn**: A PqConnection created by dbConnect().
- **statement**: An SQL string to execute.
- **immediate**: If TRUE, uses the PGsendQuery() API instead of PGprepare(). This allows to pass multiple statements and turns off the ability to pass parameters.

### 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

```r
library(DBI)
db <- dbConnect(RPostgres::Postgres())
dbWriteTable(db, "usrarrests", 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)
```
postgres-tables

Convenience functions for reading/writing DBMS tables

Description

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

Usage

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

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

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

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

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

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

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

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

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

### 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.

- **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()`.

- **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.

- **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.

- **check.names**
  - If TRUE, the default, column names will be converted to valid R identifiers.

- **temporary**
  - If TRUE, only temporary tables are considered.

- **fail_if_missing**
  - If FALSE, `dbRemoveTable()` succeeds if the table doesn’t exist.
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.

con A database connection.

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 savepoint.

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.
postgresIsTransacting

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

```r
postgresHasDefault(...)  
postgresDefault(...)  
```

Arguments

...  

Additional arguments passed on to `dbConnect()`

Examples

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

postgresIsTransacting  Return whether a transaction is ongoing

Description

Detect whether the transaction is active for the given connection. A transaction might be started with `dbBegin()` or wrapped within `DBI::dbWithTransaction()`.

Usage

```r
postgresIsTransacting(conn)
```

Arguments

`conn`  

a `PqConnection` object, produced by `DBI::dbConnect()`

Value

A boolean, indicating if a transaction is ongoing.
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)
quote

} 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,Id'
dbQuoteIdentifier(conn, x, ...)

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

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

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

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

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

## S4 method for signature 'PqConnection,SQL'
dbUnquoteIdentifier(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

```r
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 `drv = Redshift()` instead of `drv = 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

```r
Redshift()
```

### S4 method for signature 'RedshiftDriver'

```r
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**
  - `DBI::DBIDriver`. Use `Postgres()` to connect to a PostgreSQL(-ish) database or `Redshift()` to connect to an AWS Redshift cluster. Use an existing `DBI::DBIConnection` object to clone an existing connection.

- **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.
Index

bit64::integer64, 5, 15

dbAppendTable(), 7
dbAppendTable, PqConnection-method (postgres-tables), 7
dbAppendTable_PqConnection (postgres-tables), 7
dbBegin(), 11
dbBegin, PqConnection-method (postgres-transactions), 9
dbBegin_PqConnection (postgres-transactions), 9
dbBind, PqResult-method (postgres-query), 5
dbBind_PqResult (postgres-query), 5
dbClearResult, PqResult-method (postgres-query), 5
dbClearResult_PqResult (postgres-query), 5
dbCommit, PqConnection-method (postgres-transactions), 9
dbCommit_PqConnection (postgres-transactions), 9
dbConnect(), 6, 11
dbConnect, PqDriver-method (Postgres), 4
dbConnect, RedshiftDriver-method (Redshift), 14
dbConnect_PqDriver (Postgres), 4
dbConnect_RedshiftDriver (Redshift), 14
dbDisconnect, PqConnection-method (Postgres), 4
dbDisconnect_PqConnection (Postgres), 4
dbExistsTable, PqConnection, character-method (postgres-tables), 7
dbExistsTable, PqConnection, Id-method (postgres-tables), 7
dbExistsTable_PqConnection_character (postgres-tables), 7
dbExistsTable_PqConnection_Id (postgres-tables), 7
dbFetch, PqResult-method (postgres-query), 5
dbFetch_PqResult (postgres-query), 5
dbGetRowsAffected(), 6
dbHasCompleted, PqResult-method (postgres-query), 5
dbHasCompleted_PqResult (postgres-query), 5
DBI::dbConnect(), 8, 10–12
DBI::dbDataType(), 9
DBI::DBICore, 4, 14
DBI::DBIDriver, 4, 14
DBI::dbSendQuery(), 6
DBI::dbWithTransaction(), 11
dbListFields, PqConnection, character-method (postgres-tables), 7
dbListFields, PqConnection, Id-method (postgres-tables), 7
dbListFields_PqConnection_character (postgres-tables), 7
dbListFields_PqConnection_Id (postgres-tables), 7
dbListObjects, PqConnection-method (postgres-tables), 7
dbListObjects_PqConnection_ANY (postgres-tables), 7
dbListTables, PqConnection-method (postgres-tables), 7
dbListTables_PqConnection (postgres-tables), 7
dbListTables_PqConnection_ANY (postgres-tables), 7
dbQuoteIdentifier(), 8
dbQuoteIdentifier, PqConnection, character-method (quote), 13
dbQuoteIdentifier, PqConnection, Id-method (quote), 13
dbQuoteIdentifier, PqConnection, SQL-method (quote), 13
dbQuoteIdentifier_PqConnection_character (quote), 13
INDEX

dbQuoteIdentifier_PqConnection_Id (quote), 13
postgres-transactions, 9

dbQuoteIdentifier_PqConnection_SQL (quote), 13
postgresDefault (postgresHasDefault), 10

dbQuoteLiteral_PqConnection (quote), 13
postgresHasDefault, 10

dbQuoteLiteral_PqConnection (quote), 13
postgresIsTransacting, 11

dbQuoteString_PqConnection (quote), 13
postgresWaitForNotify, 12

PqConnection, 6, 8, 10–13
PqResult, 6

quote, 13

Redshift, 14

Redshift(), 4, 8, 14
RedshiftConnection-class (Redshift), 14

RedshiftDriver-class (Redshift), 14

RPostgres (RPostgres-package), 3
RPostgres-package, 3

SQL(), 8

sqlData_PqConnection-method (postgres-tables), 7

sqlData_PqConnection (postgres-tables), 7

Sys.timezone(), 5

Id, 13

Id(), 8, 9

Postgres, 4
Postgres(), 4, 8, 11, 14

postgres-query, 5

postgres-tables, 7