Package 'rpostgis'

January 6, 2016

Version 0.7

Date 2016-01-05

Title PostGIS and PostgreSQL related functions

Description This package provides additional functions to the RPostgreSQL package, mostly convenient wrappers, with some PostGIS oriented functions.

Author Mathieu Basille, David Bucklin

Maintainer David Bucklin <david.bucklin@gmail.com>

Depends raster,
rgeos,
RPostgreSQL,
sp

License GPL (>= 3)

URL http://ase-research.org/basille/rpostgis

RoxygenNote 5.0.1

R topics documented:

pgAddKey *Add key*

Description

Add a primary or foreign key to a table column.

Usage

```
pgAddKey(conn, name, colname, type = c("primary", "foreign"), reference,
  colref, display = TRUE, exec = TRUE)
```

Arguments

conn	A connection object.
name	A character string specifying a PostgreSQL table name.
colname	A character string specifying the name of the column to which the key will be assign.
type	The type of the key, either primary or foreign
reference	A character string specifying a foreign table name to which the foreign key will be associated.
colref	A character string specifying the name of the primary key in the foreign table to which the foreign key will be associated.
display	Logical. Whether to display the query (defaults to TRUE).
exec	Logical. Whether to execute the query (defaults to TRUE).

Author(s)

Mathieu Basille <basille@ase-research.org>

See Also

The PostgreSQL documentation: http://www.postgresql.org/docs/current/static/sql-altertable.html

Examples

```
pgAddKey(name = c("fla", "bli"), colname = "id", type = "foreign",
    reference = c("flu", "bla"), colref = "id", exec = FALSE)
```

pgAsDate *Converts to timestamp*

Description

Convert a date field to a timestamp with or without time zone.

Usage

```
pgAsDate(conn, name, date = "date", tz = NULL, display = TRUE,
  exec = TRUE)
```

Arguments

conn	A connection object.
name	A character string specifying a PostgreSQL table name.
date	A character string specifying the date field.
tz	A character string specifying the time zone, in "EST", "America/New_York", "EST5EDT", "-5".
display	Logical. Whether to display the query (defaults to TRUE).
exec	Logical. Whether to execute the query (defaults to TRUE).

Author(s)

Mathieu Basille <basille@ase-research.org>

See Also

The PostgreSQL documentation: http://www.postgresql.org/docs/current/static/datatype-datetime.html

Examples

```
pgAsDate(name = c("fla", "bli"), date = "date", tz = "GMT", exec = FALSE)
```

pgColumn *Add or remove a column*

Description

Add or remove a column to/from a table.

Usage

```
pgColumn(conn, name, colname, action = c("add", "drop"),
  coltype = "integer", cascade = FALSE, display = TRUE, exec = TRUE)
```

Arguments

conn	A connection object.
name	A character string specifying a PostgreSQL table name.
colname	A character string specifying the name of the column to which the key will be associated.
action	A character string specifying if the column is to be added ("add", default) or removed ("drop").
coltype	A character string indicating the type of the column, if action = "add".
cascade	Logical. Whether to drop foreign key constraints of other tables, if action = "drop".
display	Logical. Whether to display the query (defaults to TRUE).
exec	Logical. Whether to execute the query (defaults to TRUE).

Author(s)

Mathieu Basille <basille@ase-research.org>

See Also

The PostgreSQL documentation: http://www.postgresql.org/docs/current/static/sql-altertable.html

Examples

```
## Add an integer column
pgColumn(name = c("fla", "bli"), colname = "field", exec = FALSE)
## Drop a column (with CASCADE)
pgColumn(name = c("fla", "bli"), colname = "field", action = "drop",
    cascade = TRUE, exec = FALSE)
```

---

pgComment *Comment table/view/schema*

Description

Comment on a table, a view or a schema.

Usage

```
pgComment(conn, name, comment, type = c("table", "view", "schema"),
  display = TRUE, exec = TRUE)
```

Arguments

conn	A connection object.
name	A character string specifying a PostgreSQL table, view or schema name.
comment	A character string specifying the comment.
type	The type of the object to comment, either table or view
display	Logical. Whether to display the query (defaults to TRUE).
exec	Logical. Whether to execute the query (defaults to TRUE).

Author(s)

Mathieu Basille <basille@ase-research.org>

See Also

The PostgreSQL documentation: http://www.postgresql.org/docs/current/static/sql-comment.html

Examples

```
pgComment(name = c("fla", "bli"), comment = "Comment on a view.",
    type = "view", exec = FALSE)
pgComment(name = "fla", comment = "Comment on a schema.", type = "schema",
    exec = FALSE)
```

---

pgDrop *Drop table/view/schema*

Description

Drop a table, a view or a schema.

Usage

```
pgDrop(conn, name, type = c("table", "view", "schema"), ifexists = FALSE,
  cascade = FALSE, display = TRUE, exec = TRUE)
```

Arguments

conn	A connection object.
name	A character string specifying a PostgreSQL table, view or schema name.
type	The type of the object to comment, either table or view
ifexists	Do not throw an error if the table does not exist. A notice is issued in this case.
cascade	Automatically drop objects that depend on the table (such as views).
display	Logical. Whether to display the query (defaults to TRUE).
exec	Logical. Whether to execute the query (defaults to TRUE).

Author(s)

Mathieu Basille <basille@ase-research.org>

See Also

The PostgreSQL documentation: http://www.postgresql.org/docs/current/static/sql-droptable.html, http://www.postgresql.org/docs/current/static/sql-dropview.html, http://www.postgresql.org/docs/current/static/sql-dropschema.html

Examples

```
pgDrop(name = c("fla", "bli"), type = "view", exec = FALSE)
pgDrop(name = "fla", type = "schema", cascade = "TRUE", exec = FALSE)
```

pgGetBoundary *Returns bounding envelope of all combined geometries or rasters stored in a table in a PostgreSQL database.*

Description

Retrieve bounding envelope (rectangle) of all geometries or rasters in a table in Postgresql.

Usage

```
pgGetBoundary(conn, table, geom = "geom", raster = FALSE)
```

Arguments

conn	A connection object to a PostgreSQL database
table	A character string specifying a PostgreSQL schema (if necessary), and table or view name for the table holding the geometries/raster(s) (e.g., table = c("schema","table"))
geom	character, Name of the column in 'table' holding the geometry or raster object (Default = 'geom')
raster	logical, Set to TRUE if using for raster objects (Default = FALSE)

Value

SpatialPolygon

Author(s)

David Bucklin <david.bucklin@gmail.com>

Examples

```
## Not run:
library(RPostgreSQL)
drv<-dbDriver("PostgreSQL")
conn<-dbConnect(drv,dbname='dbname',host='host',port='5432',
               user='user',password='password')

pgGetBoundary(conn,c('schema','polys'),geom = 'polygon')
pgGetBoundary(conn,c('schema','rasters'),geom='rast',raster=TRUE)

## End(Not run)
```

pgGetLines *Load a linestring geometry stored in a PostgreSQL database into R.*

Description

Retrieve line geometries from a PostGIS table, and convert it to a SpatialLines or a SpatialLinesDataFrame.

Usage

```
pgGetLines(conn, table, geom = "geom", gid = NULL, other.cols = "*",
  query = NULL)
```

Arguments

conn	A connection object to a PostgreSQL database
table	A character string specifying a PostgreSQL schema (if necessary), and table or view name for the table holding the lines geometry (e.g., table = c("schema","table"))
geom	character, Name of the column in 'table' holding the geometry object (Default = 'geom')
gid	character, Name of the column in 'table' holding the ID for each line. Should be unique if additional columns of unique data are being appended. (Default = 'gid')
other.cols	character, names of additional columns from table (comma-seperated) to append to dataset (Default is all columns, NULL returns a SpatialLines object)
query	character, additional SQL to append to modify select query from table

Value

SpatialLinesDataFrame or SpatialLines

Author(s)

David Bucklin <david.bucklin@gmail.com>

Examples

```
## Not run:
library(RPostgreSQL)
drv<-dbDriver("PostgreSQL")
conn<-dbConnect(drv,dbname='dbname',host='host',port='5432',
               user='user',password='password')

pgGetLines(conn,c('schema','tablename'))
pgGetLines(conn,c('schema','roads'),geom='roadgeom',gid='road_ID',
           other.cols=NULL, query = "AND field = \'highway\'")

## End(Not run)
```

pgGetPolys *Load a polygon geometry stored in a PostgreSQL database into R.*

Description

Retrieve polygon geometries from a PostGIS table, and convert it to a SpatialPolygons or a SpatialPolygonsDataFrame.

Usage

```
pgGetPolys(conn, table, geom = "geom", gid = NULL, other.cols = "*",
  query = NULL)
```

Arguments

conn	A connection object to a PostgreSQL database
table	A character string specifying a PostgreSQL schema (if necessary), and table or view name for the table holding the polygon geometry (e.g., table = c("schema","table"))
geom	character, Name of the column in 'table' holding the geometry object (Default = 'geom')
gid	character, Name of the column in 'table' holding the ID for each polygon geometry. Should be unique if additional columns of unique data are being appended. (Default = 'gid')
other.cols	character, names of additional columns from table (comma-seperated) to append to dataset (Default is all columns, other.cols=NULL returns a SpatialPolygons object)
query	character, additional SQL to append to modify select query from table
proj	numeric, Can be set to TRUE to automatically take the SRID for the table in the database. Alternatively, the number of EPSG-specified projection of the geometry (Default is NULL, resulting in no projection.)

Value

SpatialPolygonsDataFrame or SpatialPolygons

Author(s)

David Bucklin <david.bucklin@gmail.com>

Examples

```
## Not run:
library(RPostgreSQL)
drv<-dbDriver("PostgreSQL")
conn<-dbConnect(drv,dbname='dbname',host='host',port='5432',
               user='user',password='password')

pgGetPolys(conn,c('schema','tablename'))
pgGetPolys(conn,c('schema','states'),geom='statesgeom',gid='state_ID',
           other.cols='area,population',
```

```
            query = "AND area > 1000000 ORDER BY population LIMIT 10")

## End(Not run)
```

---

pgGetPts *Retrieve point geometries*

Description

Retrieve point geometries from a PostGIS table, and convert it to a SpatialPoints or a SpatialPointsDataFrame.

Usage

```
pgGetPts(conn, table, geom = "geom", gid = NULL, other.cols = "*",
  query = NULL)
```

Arguments

conn	A connection object to a PostgreSQL database
table	A character string specifying a PostgreSQL schema (if necessary), and table or view name for the table holding the points geometry (e.g., table = c("schema","table"))
geom	The name of the point geometry column. (Default = 'geom')
gid	Name of the column in 'table' holding the ID. Should be unique if additional columns of unique data are being appended. gid=NULL (default) automatically creates a new unique ID for each row in the table.
other.cols	Names of specific columns in the table to retrieve, comma seperated in one character element (e.g. other.cols='col1,col2'. The default is to attach all columns in a SpatialPointsDataFrame. Setting other.cols=NULL will return a SpatialPoints.

Value

A Spatial(Multi)Points or a Spatial(Multi)PointsDataFrame

Author(s)

David Bucklin <david.bucklin@gmail.com>

Mathieu Basille <basille@ase-research.org>

Examples

```
## Not run:
## Retrieve a SpatialPointsDataFrame with all data from table 'schema.tablename',
with geometry in the column 'geom'
pgGetPts(conn, c('schema','tablename'))
## Return a SpatialPointsDataFrame with columns c1 & c2 as data
pgGetPts(conn, c('schema','tablename'), other.cols = 'c1,c2')
## Return a SpatialPoints, retaining id from table as rownames
pgGetPts(conn, c('schema','tablename'), gid = 'table_id', other.cols = FALSE)

## End(Not run)
```

pgGetRast *Load a raster stored in a PostgreSQL database into R.*

Description

Retrieve rasters from a PostGIS table

Usage

```
pgGetRast(conn, table, rast = "rast", digits = 9, NSEW = c(NULL, NULL,
  NULL, NULL))
```

Arguments

conn	A connection object to a PostgreSQL database
table	A character string specifying a PostgreSQL schema (if necessary), and table or view name for the table holding the raster (e.g., table = c("schema","table"))
rast	Name of the column in 'table' holding the raster object
digits	numeric, precision for detecting whether points are on a regular grid (a low number of digits is a low precision) - From rasterFromXYZ function (raster package)
NSEW	numeric, clipping box for raster with four arguments (north, south, east, west) indicating the projection-specific limits with which to clip the raster.

Value

RasterLayer

Author(s)

David Bucklin <david.bucklin@gmail.com>

Examples

```
## Not run:
library(RPostgreSQL)
drv<-dbDriver("PostgreSQL")
conn<-dbConnect(drv,dbname='dbname',host='host',port='5432',
               user='user',password='password')

pgGetRast(conn,c('schema','tablename'))
pgGetRast(conn,c('schema','DEM'),digits=9,
         NSEW=c(55,50,17,12))

## End(Not run)
```

pgIndex *CREATE INDEX*

Description

Defines a new index.

Usage

```
pgIndex(conn, name, colname, idxname, unique = FALSE, method = c("btree",
  "hash", "rtree", "gist"), display = TRUE, exec = TRUE)
```

Arguments

conn	A connection object.
name	A character string specifying a PostgreSQL table name.
colname	A character string specifying the name of the column to which the key will be associated.
idxname	A character string specifying the name of the index to be created. By default, this is the name of the table (without the schema) suffixed by _idx.
unique	Logical. Causes the system to check for duplicate values in the table when the index is created (if data already exist) and each time data is added. Attempts to insert or update data which would result in duplicate entries will generate an error.
method	The name of the method to be used for the index. Choices are "btree", "hash", "rtree", and "gist". The default method is btree.
display	Logical. Whether to display the query (defaults to TRUE).
exec	Logical. Whether to execute the query (defaults to TRUE).

Author(s)

Mathieu Basille <basille@ase-research.org>

See Also

The PostgreSQL documentation: http://www.postgresql.org/docs/current/static/sql-createindex.html; the PostGIS documentation for GiST indexes: http://postgis.net/docs/using_postgis_dbmanagement.html#id541286

Examples

```
pgIndex(name = c("fla", "bli"), colname = "wkb_geometry", method = "gist",
    exec = FALSE)
```

pgMakePts *Add a POINT or LINESTRING geometry field.*

Description

Add a new POINT or LINESTRING geometry field.

Usage

```
pgMakePts(conn, name, colname = "pts_geom", x = "x", y = "y", srid,
  index = TRUE, display = TRUE, exec = TRUE)

pgMakeStp(conn, name, colname = "stp_geom", x = "x", y = "y", dx = "dx",
  dy = "dy", srid, index = TRUE, display = TRUE, exec = TRUE)
```

Arguments

conn	A connection object.
name	A character string specifying a PostgreSQL table name.
colname	A character string specifying the name of the new geometry column.
x	The name of the x/longitude field.
y	The name of the y/latitude field.
srid	A valid SRID for the new geometry.
index	Logical. Whether to create an index on the new geometry.
display	Logical. Whether to display the query (defaults to TRUE).
exec	Logical. Whether to execute the query (defaults to TRUE).
dx	The name of the dx field (i.e. increment in x direction).
dy	The name of the dy field (i.e. increment in y direction).

Author(s)

Mathieu Basille <basille@ase-research.org>

See Also

The PostGIS documentation for ST_MakePoint: http://postgis.net/docs/ST_MakePoint.html, and for ST_MakeLine: http://postgis.net/docs/ST_MakeLine.html, which are the main functions of the call.

Examples

```
## Create a new POINT field called "pts_geom"
pgMakePts(name = c("fla", "bli"), x = "longitude", y = "latitude",
    srid = 4326, exec = FALSE)

## Create a new LINESTRING field called "stp_geom"
pgMakeStp(name = c("fla", "bli"), x = "longitude", y = "latitude",
    dx = "xdiff", dy = "ydiff", srid = 4326, exec = FALSE)
```

pgSchema *Create schema*

Description

Create a schema.

Usage

```
pgSchema(conn, name, display = TRUE, exec = TRUE)
```

Arguments

conn	A connection object.
name	A character string specifying a PostgreSQL schema name.
display	Logical. Whether to display the query (defaults to TRUE).
exec	Logical. Whether to execute the query (defaults to TRUE).

Author(s)

Mathieu Basille <basille@ase-research.org>

See Also

The PostgreSQL documentation: http://www.postgresql.org/docs/current/static/sql-createschema.html

Examples

```
pgSchema(name = "schema", exec = FALSE)
```

pgVacuum *VACUUM*

Description

Performs a VACUUM (garbage-collect and optionally analyze) on a table.

Usage

```
pgVacuum(conn, name, full = FALSE, verbose = FALSE, analyze = TRUE,
  display = TRUE, exec = TRUE)
```

Arguments

conn	A connection object.
name	A character string specifying a PostgreSQL table name.
full	Logical. Whether to perform a "full" vacuum, which can reclaim more space, but takes much longer and exclusively locks the table.
verbose	Logical. Whether to print a detailed vacuum activity report for each table.
analyze	Logical. Whether to update statistics used by the planner to determine the most efficient way to execute a query (default to TRUE).
display	Logical. Whether to display the query (defaults to TRUE).
exec	Logical. Whether to execute the query (defaults to TRUE).

Author(s)

Mathieu Basille <basille@ase-research.org>

See Also

The PostgreSQL documentation: http://www.postgresql.org/docs/current/static/sql-vacuum.html

Examples

```
pgVacuum(name = c("fla", "bli"), full = TRUE, exec = FALSE)
```

---

rpostgis *PostGIS and PostgreSQL functions*

Description

rpostgis

Details

This package provides additional functions to the RPostgreSQL package, mostly convenient wrappers to PostgreSQL queries, with some PostGIS oriented functions. For a list of documented functions, use `library(help = "rpostgis")`

Author(s)

Mathieu Basille <basille@ase-research.org>

Index