

Package ‘rpostgis’

July 25, 2016

Version 0.9

Date 2016-07-20

Title PostGIS and PostgreSQL related functions

Description This package provides general usage functions for PostgreSQL and PostGIS, including geometry/raster import into R and geometry export from R to PostgreSQL.

Depends R (>= 3.3.0),
RPostgreSQL

Imports methods,
raster,
rgdal,
rgeos,
sp,
stats

Suggests wkb

License GPL (>= 3)

LazyData true

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

RoxygenNote 5.0.1

R topics documented:

| | |
|---------------------------|----|
| pgAddKey | 2 |
| pgAsDate | 3 |
| pgColumn | 3 |
| pgColumnInfo | 4 |
| pgComment | 5 |
| pgDrop | 6 |
| pgGetBoundary | 7 |
| pgGetPts | 8 |
| pgGetRast | 9 |
| pgIndex | 10 |
| pgInsert | 11 |
| pgInsertizeGeom | 12 |
| pgMakePts | 15 |
| pgPostGIS | 16 |
| pgSchema | 17 |

| | |
|--------------------|----|
| pgSRID | 17 |
| pgVacuum | 18 |
| rpostgis | 19 |

| | |
|--------------|-----------|
| Index | 20 |
|--------------|-----------|

| | |
|----------|----------------|
| pgAddKey | <i>Add key</i> |
|----------|----------------|

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@ufl.edu>

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 | <i>Converts to timestamp</i> |
|----------|------------------------------|

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@ufl.edu>

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 | <i>Add or remove a column</i> |
|----------|-------------------------------|

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@uf1.edu>

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)
```

pgColumnInfo

Get information about columns in a PostgreSQL table.

Description

Get information about columns in a PostgreSQL table.

Usage

```
pgColumnInfo(conn, name, allinfo = FALSE)
```

Arguments

| | |
|---------|--|
| conn | A connection object to a PostgreSQL database |
| name | A character string specifying a PostgreSQL schema (if necessary), and table or view name geometry (e.g., name = c("schema","table")) |
| allinfo | logical, Get all information on table? Default is column names, types, nullable, and maximum length of character columns |

Value

data frame

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')
pgColumnInfo(conn, c("schema", "table"))

## End(Not run)
```

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@uf1.edu>

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 | <i>Drop table/view/schema</i> |
|--------|-------------------------------|

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@uf1.edu>

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, name, geom = "geom")
```

Arguments

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

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')

## End(Not run)
```

pgGetPts

*Load a PostGIS geometry in a PostgreSQL table/view into R.***Description**

Retrieve point, linestring, or polygon geometries from a PostGIS table/view, and convert it to an R 'sp' object (Spatial* or Spatial*DataFrame)

Usage

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

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

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

Arguments

| | |
|------------|--|
| conn | A connection object to a PostgreSQL database |
| name | A character string specifying a PostgreSQL schema and table/view name holding the geometry (e.g., 'name = c("schema","table")') |
| geom | The name of the geometry column. (Default = 'geom') |
| gid | Name of the column in 'name' holding the IDs. 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 'sp' object. |
| other.cols | Names of specific columns in the table to retrieve, comma separated in one character element (e.g. other.cols='col1,col2'. The default is to attach all columns in a Spatial*DataFrame. Setting other.cols=NULL will return a Spatial-only object (no data). |
| query | character, additional SQL to append to modify select query from table. Must begin with "AND ..."; see below for examples. |

Value

Spatial(Multi)PointsDataFrame or Spatial(Multi)Points

SpatialLinesDataFrame or SpatialLines

SpatialPolygonsDataFrame or SpatialPolygons

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)
## 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)
## 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)
```

pgGetRast

Load a raster stored in a PostgreSQL database into R.

Description

Retrieve rasters from a PostGIS table

Usage

```
pgGetRast(conn, name, rast = "rast", digits = 9, boundary = NULL)
```

Arguments

| | |
|------|--|
| conn | A connection object to a PostgreSQL database |
| name | A character string specifying a PostgreSQL schema (if necessary), and table or view name for the table holding the raster (e.g., name = c("schema","table")) |
| rast | Name of the column in 'name' 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) |
| boundary | sp object or numeric. A Spatial* object, whose bounding box will be used to select the part of the raster to import. Alternatively, four numbers (e.g. c(north, south, east, west)) indicating the projection-specific limits with which to clip the raster. NULL (default) will return the full 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,
          boundary=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@ufl.edu>

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)
```

| | |
|----------|--|
| pgInsert | <i>Inserts data from a pgInsertize* object into a PostgreSQL table</i> |
|----------|--|

Description

This function takes a `pgi` list output object from `pgInsertize` or `pgInsertizeGeom` and performs the database insert (and table creation, if specified in the previous functions) on the database. If `create.table` or `force.match` were not specified in the `pgInsertize*` statement, the table to insert into should be specified in `name` in this function. If a new table is created but the data insert statement fails, the new table is dropped from the database (a message will be given).

Usage

```
pgInsert(conn, pgi, name = NULL, encoding = NULL)
```

Arguments

| | |
|----------|---|
| conn | A connection object to a PostgreSQL database |
| pgi | The output PostgreSQL insert object (<code>pgi</code>) created by <code>pgInsertize()</code> or <code>pgInsertizeGeom()</code> |
| name | character strings specifying a PostgreSQL schema and table name to insert into (e.g., <code>name = c("schema","table")</code>). If table was specified in the <code>pgInsertize*</code> through <code>create.table</code> or <code>force.match</code> , leave this <code>NULL</code> . |
| encoding | Character vector of length 2, containing the from/to encodings for the data (as in the function <code>iconv</code>). For example, if the dataset contain certain latin characters (e.g., accent marks), and the database is in UTF-8, use <code>encoding = c("latin1","UTF-8")</code> . Left <code>NULL</code> , no conversion will be done. |

Value

DBIResult

Author(s)

David Bucklin <david.bucklin@gmail.com>

Examples

```
library(sp)
data(meuse)
coords <- SpatialPoints(meuse[, c("x", "y")])
spdf<- SpatialPointsDataFrame(coords, meuse)

#format data for insert
pgi<-pgInsertizeGeom(spdf,geom="point")

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

# insert data in database table (note that an error will be given if
# all insert columns do not match exactly to database table columns)
pgInsert(conn,pgi=pgi,name=c("schema","meuse_data"))

## End(Not run)
```

pgInsertizeGeom

Format R data objects (data frames, spatial data frames, or spatial-only objects) for insert into a PostgreSQL table (for use with pgInsert).

Description

These functions take an R sp object (Spatial* or Spatial*DataFrame; for pgInsertizeGeom) or data frame (for pgInsertize) and return a pgi list object, which is used in the function pgInsert to insert rows of the object into the database table. (Note that these functions do not do any modification of the database, it only prepares the data for insert.) The entire data frame is prepared by default, unless force.match specifies a database table (along with a database connection conn), in which case the R column names are compared to the force.match column names, and only exact matches are formatted to be inserted. A new database table can also be prepared to be created (if so, the actual table is created in pgInsert) using the create.table argument. If new.id is specified, a new sequential integer field is added to the data frame. For Spatial*-only objects (no data frame), a new.id is created by default with name "gid". For pgInsertizeGeom, if the R package wkB is installed, this function uses writeWKB to translate the geometries (faster for large datasets), otherwise the rgeos function writeWKT is used.

Usage

```
pgInsertizeGeom(data.obj, geom = "geom", create.table = NULL,
  force.match = NULL, conn = NULL, new.id = NULL, alter.names = TRUE)

pgInsertize(data.obj, create.table = NULL, force.match = NULL,
  conn = NULL, new.id = NULL, alter.names = TRUE)

## S3 method for class 'pgi'
print(pgi)
```

Arguments

| | |
|---------------------------|---|
| <code>data.obj</code> | A Spatial* or Spatial*DataFrame, or data frame for pgInsertize. |
| <code>geom</code> | character string, the name of geometry column in the database table. (existing or to be created; defaults to 'geom') |
| <code>create.table</code> | character, schema and table of the PostgreSQL table to create (actual table creation will be done in later in pgInsert(.).) Column names will be converted to PostgreSQL-compliant names. Default is NULL (no new table created). |
| <code>force.match</code> | character, schema and table of the PostgreSQL table to compare columns of data frame with. If specified, only columns in the data frame that exactly match the database table will be kept, and reordered to match the database table. If NULL, all columns will be kept in the same order given in the data frame. |
| <code>conn</code> | A database connection (if a table is given in for "force.match" parameter) |
| <code>new.id</code> | character, name of a new sequential integer ID column to be added to the table. (for spatial objects without data frames, this column is created even if left NULL and defaults to the name 'gid') |
| <code>alter.names</code> | Logical, whether to make database column and table names DB-compliant (remove special characters). Default is TRUE. (This will need to be set to FALSE if matching to non-standard names in an existing database table using the <code>force.match</code> setting.) |
| <code>pgi</code> | A list of class <code>pgi</code> , output from the <code>pgInsertize()</code> or <code>pgInsertizeGeom()</code> functions from the <code>rpostgis</code> package. |

Value

`pgi` A list containing four character strings- a list containing four character strings- (1) `in.table`, the table name which will be created or inserted into, if specified by either `create.table` or `force.match` (else NULL) (2) `db.new.table`, the SQL statement to create the new table, if specified in `create.table` (else NULL), (3) `db.cols.insert`, a character string of the database column names to insert into, and (4) `insert.data`, a character string of the data to insert. See examples for usage within the `pgInsert` function.

Author(s)

David Bucklin <david.bucklin@gmail.com>

Examples

```
library(sp)
data(meuse)
```

```

coords <- SpatialPoints(meuse[, c("x", "y")])
spdf<- SpatialPointsDataFrame(coords, meuse)

#format data for insert
pgi.new<-pgInsertizeGeom(spdf,geom="point_geom",create.table=c("schema","table"),new.id="pt_gid")
print(pgi.new)

## Not run:

library(RPostgreSQL)
drv<-dbDriver("PostgreSQL")
conn<-dbConnect(drv,dbname='dbname',host='host',port='5432',
                user='user',password='password')

# insert data in database table (note that an error will be given if all
# insert columns do not have exactly matching database table columns)
pgInsert(conn,pgi=pgi.new)

# Inserting into existing table
pgi.existing<-pgInsertizeGeom(spdf,geom="point_geom",force.match=c("schema","table"),conn=conn)
# A warning message is given, since the "dist.m" column is not found in the database table
# (it was changed to "dist_m" in pgi.new to make name DB-compliant).
# All other columns are prepared for insert.
print(pgi.existing)

pgInsert(conn,pgi=pgi.existing)

## End(Not run)

## Not run:

#format regular (non-spatial) data frame for insert using pgInsertize

#connect to database
library(RPostgreSQL)
drv<-dbDriver("PostgreSQL")
conn<-dbConnect(drv,dbname='dbname',host='host',port='5432',
                user='user',password='password')

## End(Not run)

data<-data.frame(a=c(1,2,3),b=c(4,NA,6),c=c(7,'text',9))

#format non-spatial data frame for insert
values<-pgInsertize(data.obj=data)

## Not run:
# insert data in database table (note that an error will be given if all insert columns
# do not match exactly to database table columns)
pgInsert(conn,pgi=values,name=c("schema","table"))

##
#run with forced matching of database table column names
values<-pgInsertize(data.obj=data,force.match=c("schema","table"),conn=conn)

pgInsert(conn,pgi=values)

```

```
## End(Not run)
```

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@ufl.edu>

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)
```

pgPostGIS

Check and create PostGIS extension.

Description

The function checks for the availability of the PostGIS extension, and if it is available, but not installed, install it. Additionnaly, can also install Topology, Tiger Geocoder and SFCGAL extensions.

Usage

```
pgPostGIS(conn, topology = FALSE, tiger = FALSE, sfcgal = FALSE,  
          display = TRUE, exec = TRUE)
```

Arguments

| | |
|----------|---|
| conn | A connection object (required, even if exec = FALSE). |
| topology | Logical. Whether to check/install the Topology extension. |
| tiger | Logical. Whether to check/install the Tiger Geocoder extension. |
| sfcgal | Logical. Whether to check/install the SFCGAL extension. |
| display | Logical. Whether to display the query (defaults to TRUE). |
| exec | Logical. Whether to execute the query (defaults to TRUE). |

Value

TRUE if PostGIS is installed.

Author(s)

Mathieu Basille <basille@uf1.edu>

Examples

```
## 'exec = FALSE' does not install any extension, but nevertheless  
## check for available and installed extensions:  
## Not run:  
  pgPostGIS(con, topology = TRUE, tiger = TRUE, sfcgal = TRUE,  
            exec = FALSE)  
  
## End(Not run)
```

| | |
|----------|---------------------------------|
| pgSchema | <i>Check and create schema.</i> |
|----------|---------------------------------|

Description

Checks the existence, and if necessary, creates a schema.

Usage

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

Arguments

| | |
|---------|---|
| conn | A connection object (required, even if exec = FALSE). |
| 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). |

Value

TRUE if the schema exists (whether it was already available or was just created).

Author(s)

Mathieu Basille <basille@ufl.edu>

See Also

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

Examples

```
## Not run:
  pgSchema(name = "schema", exec = FALSE)

## End(Not run)
```

| | |
|--------|--|
| pgSRID | <i>Find the matching PostGIS SRID for a CRS object (or create a new SRID if not found)</i> |
|--------|--|

Description

This function takes ‘CRS’-class object and a PostgreSQL database connection (with PostGIS extension), and returns the matching SRID(s) for that CRS. If a match is not found, a new entry can be created in the PostgreSQL ‘spatial_ref_sys’ table using the parameters specified by the CRS. New entries will be created with auth_name = ‘rpostgis_custom’.

Usage

```
pgSRID(CRS, conn, create = FALSE, new.srid = NULL)
```

Arguments

| | |
|----------|--|
| CRS | CRS object, created through a call to 'CRS()' from library 'sp'. |
| conn | A connection object to a PostgreSQL database |
| create | Logical. If no matching SRID is found, should a new SRID be created? User must have write access on spatial_ref_sys table. |
| new.srid | integer. Optional SRID to give to a newly created SRID. If left NULL (default), the next open value of 'srid' in 'spatial_ref_sys' between 880000 and 890000 will be used. |

Value

SRID code (integer)

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')

crs<-CRS("+proj=longlat")
pgSRID(crs,conn)

crs2<-CRS(paste("+proj=stere +lat_0=52.15616055555555",
               "+lon_0=5.38763888888889 +k=0.999908 +x_0=155000 +y_0=463000 +ellps=bessel",
               "+towgs84=565.237,50.0087,465.658,-0.406857,0.350733,-1.87035,4.0812",
               "+units=m"))
pgSRID(crs2,conn,create=TRUE)

## End(Not run)
```

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@ufl.edu>

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@ufl.edu>

Index

`pgAddKey`, [2](#)
`pgAsDate`, [3](#)
`pgColumn`, [3](#)
`pgColumnInfo`, [4](#)
`pgComment`, [5](#)
`pgDrop`, [6](#)
`pgGet...` (`pgGetPts`), [8](#)
`pgGetBoundary`, [7](#)
`pgGetLines` (`pgGetPts`), [8](#)
`pgGetPolys` (`pgGetPts`), [8](#)
`pgGetPts`, [8](#)
`pgGetRast`, [9](#)
`pgIndex`, [10](#)
`pgInsert`, [11](#)
`pgInsertize` (`pgInsertizeGeom`), [12](#)
`pgInsertizeGeom`, [12](#)
`pgMakePts`, [15](#)
`pgMakeStp` (`pgMakePts`), [15](#)
`pgPostGIS`, [16](#)
`pgSchema`, [17](#)
`pgSRID`, [17](#)
`pgVacuum`, [18](#)
`print.pgi` (`pgInsertizeGeom`), [12](#)

`rpostgis`, [19](#)
`rpostgis-package` (`rpostgis`), [19](#)