

Package ‘rpostgis’

September 12, 2016

Version 1.0.2

Date 2016-09-08

Title R Interface to a 'PostGIS' Database

Description Provides additional functions to the 'RPostgreSQL' package to interface R with a 'PostGIS'-enabled database, as well as convenient wrappers to common 'PostgreSQL' queries.

Depends R (>= 3.3.0),
RPostgreSQL,
DBI

Imports methods,
raster,
rgeos,
sp,
stats

Suggests rgdal,
wkb

License GPL (>= 3)

LazyData true

URL <https://github.com/mablalab/rpostgis>

BugReports <https://github.com/mablalab/rpostgis/issues>

RoxygenNote 5.0.1

R topics documented:

| | |
|-------------------------|----|
| dbAddKey | 2 |
| dbAsDate | 3 |
| dbColumn | 4 |
| dbComment | 5 |
| dbDrop | 6 |
| dbIndex | 7 |
| dbSchema | 8 |
| dbTableInfo | 9 |
| dbVacuum | 9 |
| pgGetBoundary | 10 |

| | |
|----------------------|----|
| pgGetGeom | 11 |
| pgGetRast | 12 |
| pgInsert | 13 |
| pgListGeom | 15 |
| pgMakePts | 16 |
| pgPostGIS | 17 |
| pgSRID | 18 |
| rpostgis | 19 |

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

| | |
|----------|-----------------|
| dbAddKey | <i>Add key.</i> |
|----------|-----------------|

Description

Add a primary or foreign key to a table column.

Usage

```
dbAddKey(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). |

Value

TRUE if the key was successfully added.

Author(s)

Mathieu Basille <basille@ufl.edu>

See Also

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

Examples

```
## examples use a dummy connection from DBI package
conn<-DBI::ANSI()
dbAddKey(conn, name = c("fla", "bli"), colname = "id", type = "foreign",
  reference = c("flu", "bla"), colref = "id", exec = FALSE)
```

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

Description

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

Usage

```
dbAsDate(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). |

Value

If `exec = TRUE`, returns TRUE if the conversion was successful.

Author(s)

Mathieu Basille <basille@uf1.edu>

See Also

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

Examples

```
## Example uses a dummy connection from DBI package
conn <- DBI::ANSI()
dbAsDate(conn, name = c("schema", "table"), date = "date", tz = "GMT",
  exec = FALSE)
```

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

Description

Add or remove a column to/from a table.

Usage

```
dbColumn(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 |
| 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). |

Value

TRUE if the column was successfully added or removed.

Author(s)

Mathieu Basille <basille@ufl.edu>

See Also

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

Examples

```
## examples use a dummy connection from DBI package
conn<-DBI::ANSI()
## Add an integer column
dbColumn(conn, name = c("fla", "bli"), colname = "field", exec = FALSE)
## Drop a column (with CASCADE)
dbColumn(conn, name = c("fla", "bli"), colname = "field", action = "drop",
  cascade = TRUE, exec = FALSE)
```

| | |
|-----------|-----------------------------------|
| dbComment | <i>Comment table/view/schema.</i> |
|-----------|-----------------------------------|

Description

Comment on a table, a view or a schema.

Usage

```
dbComment(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). |

Value

TRUE if the comment was successfully applied.

Author(s)

Mathieu Basille <basille@ufl.edu>

See Also

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

Examples

```
## examples use a dummy connection from DBI package
conn<-DBI::ANSI()
dbComment(conn, name = c("fla", "bli"), comment = "Comment on a view.",
  type = "view", exec = FALSE)
dbComment(conn, name = "fla", comment = "Comment on a schema.", type = "schema",
  exec = FALSE)
```

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

Description

Drop a table, a view or a schema.

Usage

```
dbDrop(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). |

Value

TRUE if the table/view/schema was successfully dropped.

Author(s)

Mathieu Basille <basille@ufl.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

```
## examples use a dummy connection from DBI package  
conn<-DBI::ANSI()  
dbDrop(conn, name = c("fla", "bli"), type = "view", exec = FALSE)  
dbDrop(conn, name = "fla", type = "schema", cascade = "TRUE", exec = FALSE)
```

dbIndex*Create an index.*

Description

Defines a new index on a PostgreSQL table.

Usage

```
dbIndex(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 <code>_idx</code> . |
| 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", although "gist" should be the index of choice for Post GIS spatial types (geometry, geography, raster). |
| display | Logical. Whether to display the query (defaults to TRUE). |
| exec | Logical. Whether to execute the query (defaults to TRUE). |

Value

TRUE if the index was successfully created.

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

```
## examples use a dummy connection from DBI package  
conn<-DBI::ANSI()  
dbIndex(conn,name = c("fla", "bli"), colname = "geom", method = "gist",  
  exec = FALSE)
```

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

Description

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

Usage

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

Arguments

| | |
|---------|--|
| conn | A connection object (required, even if <code>exec = FALSE</code>). |
| name | A character string specifying a PostgreSQL schema name. |
| display | Logical. Whether to display the query (defaults to <code>TRUE</code>). |
| exec | Logical. Whether to execute the query (defaults to <code>TRUE</code>). Note: if <code>exec = FALSE</code> , the function still checks the existence of the schema, but does not create it if it does not exist. |

Value

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

Author(s)

Mathieu Basille <basille@uf1.edu>

See Also

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

Examples

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

`dbTableInfo`*Get information about table columns.*

Description

Get information about columns in a PostgreSQL table.

Usage

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

Arguments

| | |
|----------------------|--|
| <code>conn</code> | A connection object to a PostgreSQL database. |
| <code>name</code> | A character string specifying a PostgreSQL schema (if necessary), and table or view name geometry (e.g., <code>name = c("schema", "table")</code>). |
| <code>allinfo</code> | 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 <dbucklin@ufl.edu>

Examples

```
## Not run:  
dbTableInfo(conn, c("schema", "table"))  
  
## End(Not run)
```

`dbVacuum`*Vacuum.*

Description

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

Usage

```
dbVacuum(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). |

Value

TRUE if query is successfully executed.

Author(s)

Mathieu Basille <basille@ufl.edu>

See Also

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

Examples

```
## examples use a dummy connection from DBI package
conn<-DBI::ANSI()
dbVacuum(conn, name = c("fla", "bli"), full = TRUE, exec = FALSE)
```

pgGetBoundary

Retrieve bounding envelope of geometries or rasters.

Description

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

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 <dbucklin@ufl.edu>

Examples

```
## Not run:
pgGetBoundary(conn, c("schema", "polys"), geom = "polygon")
pgGetBoundary(conn, c("schema", "rasters"), geom = "rast")

## End(Not run)
```

| | |
|-----------|---|
| pgGetGeom | <i>Load a PostGIS geometry from a PostgreSQL table/view into R.</i> |
|-----------|---|

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

```
pgGetGeom(conn, name, geom = "geom", gid = NULL, other.cols = TRUE,
  clauses = 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, in a character vector (e.g. other.cols=c("col1","col2").) The default (other.cols = TRUE) is to attach all columns in a Spatial*DataFrame. Setting other.cols=FALSE will return a Spatial-only object (no data frame). |
| clauses | character, additional SQL to append to modify select query from table. Must begin with an SQL clause (e.g., "WHERE ...", "ORDER BY ...", "LIMIT ..."); see below for examples. |

Value

sp-class (SpatialPoints*, SpatialMultiPoints*, SpatialLines*, or SpatialPolygons*)

Author(s)

David Bucklin <dbucklin@ufl.edu>

Mathieu Basille <basille@ufl.edu>

Examples

```
## Not run:
## Retrieve a Spatial*DataFrame with all data from table
## 'schema.tablename', with geometry in the column 'geom'
pgGetGeom(conn, c("schema", "tablename"))
## Return a Spatial*DataFrame with columns c1 & c2 as data
pgGetGeom(conn, c("schema", "tablename"), other.cols = c("c1", "c2"))
## Return a Spatial*-only (no data frame),
## retaining id from table as rownames
pgGetGeom(conn, c("schema", "tablename"), gid = "table_id",
  other.cols = FALSE)
## Return a Spatial*-only (no data frame),
## retaining id from table as rownames and with a subset of the data
pgGetGeom(conn, c("schema", "roads"), geom = "roadgeom", gid = "road_ID",
  other.cols = FALSE, clauses = "WHERE field = 'highway'")

## End(Not run)
```

pgGetRast

*Load raster from PostGIS database.***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 <dbucklin@ufl.edu>

Examples

```
## Not run:
pgGetRast(conn, c("schema", "tablename"))
pgGetRast(conn, c("schema", "DEM"), digits = 9, boundary = c(55,
  50, 17, 12))

## End(Not run)
```

pgInsert

Inserts data into a PostgreSQL table.

Description

This function takes a take an R sp object (Spatial* or Spatial*DataFrame), or a regular data frame, and performs the database insert (and table creation, when the table doesn't exist) on the database.

Usage

```
pgInsert(conn, name, data.obj, geom = "geom", partial.match = FALSE,
  overwrite = FALSE, new.id = NULL, alter.names = TRUE, encoding = NULL,
  return.pgi = FALSE)

## S3 method for class 'pgi'
print(x, ...)
```

Arguments

| | |
|---------------|---|
| conn | A connection object to a PostgreSQL database |
| name | Character, schema and table of the PostgreSQL table to insert into. If not already existing, the table will be created. If the table already exists, the function will check if all R data frame columns match database columns, and if so, do the insert. If not, the insert will be aborted. The argument <code>partial.match</code> allows for inserts with only partial matches of data frame and database column names, and <code>overwrite</code> allows for overwriting the existing database table. |
| data.obj | A Spatial* or Spatial*DataFrame, or data frame |
| geom | character string. For Spatial* datasets, the name of geometry column in the database table. (existing or to be created; defaults to "geom"). |
| partial.match | Logical; allow insert on partial column matches between data frame and database table. If TRUE, columns in R data frame will be compared with the existing database table name. Columns in the data frame that exactly match the database table will be inserted into the database table. |
| overwrite | Logical; if true, a new table (name) will overwrite the existing table (name) in the database. |

| | |
|--------------------------|--|
| <code>new.id</code> | Character, name of a new sequential integer ID column to be added to the table for insert (for spatial objects without data frames, this column is created even if left NULL and defaults to the name "gid"). If <code>partial.match = TRUE</code> and the column does not exist in the database table, it will be discarded. |
| <code>alter.names</code> | Logical, whether to make database column names DB-compliant (remove special characters). Default is TRUE. (This should to be set to FALSE to match to non-standard names in an existing database table.) |
| <code>encoding</code> | 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 NULL, no conversion will be done. |
| <code>return.pgi</code> | Whether to return a formatted list of insert parameters (i.e., a <code>pgi</code> object; see function details.) |
| <code>x</code> | A list of class <code>pgi</code> |
| <code>...</code> | Further arguments not used. |

Details

If `new.id` is specified, a new sequential integer field is added to the data frame for insert. For Spatial*-only objects (no data frame), a `new.id` is created by default with name "gid".

If the R package `wkb` is installed, this function will use `writeWKB` for certain datasets (non-Multi types, non-`Linestring`), which is faster for large datasets. In all other cases the `rgeos` function `writeWKT` is used.

In the event of function or database error, the database uses `ROLLBACK` to revert to the previous state.

On database errors, or if the user specifies `return.pgi = TRUE`, the function will return a `pgi` object (see next paragraph). This object can be re-used as the `data.obj` in `pgInsert`; (e.g., when inserting the exact same data into tables in two separate tables or databases). If `return.pgi = FALSE` (default), the function will return TRUE, indicating successful insert.

`pgi` objects are a list containing four character strings: (1) `in.table`, the table name which will be created or inserted into (2) `db.new.table`, the SQL statement to create the new table, (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.

Value

Returns TRUE if the insertion was successful, or a `pgi` object if specified, or in the case of database error.

Author(s)

David Bucklin <dbucklin@ufl.edu>

Examples

```
## Not run:
library(sp)
data(meuse)
coords <- SpatialPoints(meuse[, c("x", "y")])
spdf <- SpatialPointsDataFrame(coords, meuse)
```

```

## Insert data in new database table
pgInsert(conn, name = c("public", "meuse_data"), data.obj = spdf)

## The same command will insert into already created table (if all R
## columns match)
pgInsert(conn, name = c("public", "meuse_data"), data.obj = spdf)

## If not all database columns match, need to use partial.match = TRUE
colnames(spdf@data)[4] <- "cu"
pgInsert(conn, name = c("public", "meuse_data"), data.obj = spdf,
  partial.match = TRUE)

## End(Not run)

```

pgListGeom

List geometries.

Description

List all geometries in a PostGIS database.

Usage

```
pgListGeom(conn, display = TRUE, exec = TRUE)
```

Arguments

| | |
|---------|---|
| conn | A PostgreSQL database connection. |
| display | Logical. Whether to display the query (defaults to TRUE). |
| exec | Logical. Whether to execute the query (defaults to TRUE). |

Value

If `exec = TRUE`, a data frame with schema, table, geometry column, and geometry type.

Author(s)

David Bucklin <dbucklin@ufl.edu>

Examples

```

## Not run:
pgListGeom(conn)

## 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 = "geom", x = "x", y = "y", srid,  
          index = TRUE, display = TRUE, exec = TRUE)
```

```
pgMakeStp(conn, name, colname = "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). |

Value

If `exec = TRUE`, returns TRUE if the geometry field was successfully created.

Author(s)

Mathieu Basille <basille@uf1.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

```
## Examples use a dummy connection from DBI package
conn <- DBI::ANSI()

## Create a new POINT field called 'pts_geom'
pgMakePts(conn, name = c("schema", "table"), colname = "pts_geom",
  x = "longitude", y = "latitude", srid = 4326, exec = FALSE)

## Create a new LINESTRING field called 'stp_geom'
pgMakeStp(conn, name = c("schema", "table"), colname = "stp_geom",
  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. Additionally, 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. Will also install extensions "fuzzystrmatch", "address_standardizer", and "address_standardizer_data_us" if all are available. |
| 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@ufl.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)
```

pgSRID

Find (or create) PostGIS SRID based on CRS object.

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'`, with the default value being the next open value between 880001-889999 (a different SRID value can be entered if desired.)

Usage

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

Arguments

| | |
|--------------------------|--|
| <code>conn</code> | A connection object to a PostgreSQL database. |
| <code>crs</code> | CRS object, created through a call to CRS . |
| <code>create.srid</code> | Logical. If no matching SRID is found, should a new SRID be created? User must have write access on <code>spatial_ref_sys</code> table. |
| <code>new.srid</code> | Integer. Optional SRID to give to a newly created SRID. If left NULL (default), the next open value of <code>srid</code> in <code>spatial_ref_sys</code> between 880001 and 889999 will be used. |

Value

SRID code (integer).

Author(s)

David Bucklin <dbucklin@ufl.edu>

Examples

```
## Not run:
drv <- dbDriver("PostgreSQL")
conn <- dbConnect(drv, dbname = "dbname", host = "host", port = "5432",
  user = "user", password = "password")
(crs <- CRS("+proj=longlat"))
pgSRID(conn, crs)
(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(conn, crs2, create.srid = TRUE)  
  
## End(Not run)
```

rpostgis

R interface to a PostGIS database.

Description

This package provides additional functions to the RPostgreSQL package to interface R with a PostGIS-enabled database, as well as convenient wrappers to common PostgreSQL queries. For a list of documented functions, use `library(help = "rpostgis")`.

Details

A typical session starts by establishing the connection to a working PostgreSQL database:

```
library(rpostgis) con <- dbConnect("PostgreSQL", dbname = <dbname>, host = <host>, user =  
<user>, password = <password>)
```

For example, this could be:

```
con <- dbConnect("PostgreSQL", dbname = "rpostgis", host = "localhost", user = "postgres", pass-  
word = "postgres")
```

The next step typically involves checking if PostGIS was installed on the working database, and if not try to install it:

```
pgPostGIS(con)
```

The function should return TRUE for all pg- functions to work.

Finally, at the end of an interactive session, the connection to the database should be closed:

```
dbDisconnect(con)
```

Author(s)

Mathieu Basille (<basille@ufl.edu>) and David Bucklin (<dbucklin@ufl.edu>)

Index

CRS, [18](#)

dbAddKey, [2](#)

dbAsDate, [3](#)

dbColumn, [4](#)

dbComment, [5](#)

dbDrop, [6](#)

dbIndex, [7](#)

dbSchema, [8](#)

dbTableInfo, [9](#)

dbVacuum, [9](#)

pgGetBoundary, [10](#)

pgGetGeom, [11](#)

pgGetRast, [12](#)

pgInsert, [13](#)

pgListGeom, [15](#)

pgMakePts, [16](#)

pgMakeStp (pgMakePts), [16](#)

pgPostGIS, [17](#)

pgSRID, [18](#)

print.pgi (pgInsert), [13](#)

rpostgis, [19](#)

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