Package 'rpostgis'

August 5, 2016

Version 0.10

Date 2016-08-01

Title 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.

Depends R (>= 3.3.0),
RPostgreSQL,
DBI

Imports methods,
raster,
rgeos,
sp,
stats

Suggests rgdal,
wkb

License GPL (>= 3)

LazyData true

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

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
pgGetPts 11
pgGetRast 12

pgInsert 13
pgInsertizeGeom 15
pgListGeomTables 17
pgMakePts 18
pgPostGIS 19
pgSRID 20
rpostgis 21

Index **22**

`dbAddKey` *Add key.*

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 *Converts to timestamp.*

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

TRUE if the conversion was successful.

Author(s)

Mathieu Basille <basille@ufl.edu>

See Also

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

Examples

```
## examples use a dummy connection from DBI package
conn<-DBI::ANSI()
dbAsDate(conn, name = c("fla", "bli"), date = "date", tz = "GMT", exec = FALSE)
```

dbColumn *Add or remove a column.*

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 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).

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 *Comment table/view/schema.*

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 *Drop table/view/schema.*

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 _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", 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 *Check and create schema.*

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 `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`). Note: if `exec = FALSE`, the function still checks the existence of the schema, but does not create it if it does not exists.

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:
    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

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 <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).

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.*

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)
```

---

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 = "*",
  clauses = NULL)

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

pgGetPolys(conn, name, geom = "geom", gid = NULL, other.cols = "*",
  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, comma seperated 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).
clauses	character, additional SQL to append to modify select query from table. Must begin with and SQL clause (e.g., "WHERE ...", "ORDER BY ...", "LIMIT ..."); see below for examples.

Value

Spatial(Multi)PointsDataFrame or Spatial(Multi)Points

SpatialLinesDataFrame or SpatialLines

SpatialPolygonsDataFrame or SpatialPolygons

Author(s)

David Bucklin <dbucklin@ufl.edu>

Mathieu Basille <basille@ufl.edu>

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:
pgGetLines(conn, c("schema", "tablename"))
pgGetLines(conn, c("schema", "roads"), geom = "roadgeom", gid = "road_ID",
    other.cols = NULL, clauses  = "WHERE field = 'highway'")

## End(Not run)
## Not run:
pgGetPolys(conn, c("schema", "tablename"))
pgGetPolys(conn, c("schema", "states"), geom = "statesgeom",
    gid = "state_ID", other.cols = "area,population",
    clauses  = "WHERE area > 1000000 ORDER BY population LIMIT 10")

## End(Not run)
```

---

pgGetRast *Load raster from DB.*

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 spatial 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)
```

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 partial.match allows for inserts with only partial matches of data frame and database column names, and overwrite 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 an 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.
new.id	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). If partial.match = TRUE and otherwise it will be discarded.
alter.names	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.
encoding	Character vector of length 2, containing the from/to encodings for the data (as in the function iconv). For example, if the dataset contain certain latin characters (e.g., accent marks), and the database is in UTF-8, use encoding = c("latin1","UTF-8"). Left NULL, no conversion will be done.

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.

Value

DBIResult

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)
```

---

pgInsertizeGeom *Format R data objects for insert into a PostgreSQL table.*

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 can be 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 function pgInsert is a wrapper around these functions, so pgInsertize* should only be used in situations where data preparation and insert need to be seperated.

Usage

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

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

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

Arguments

data.obj	A Spatial* or Spatial*DataFrame, or data frame for pgInsertize.
geom	character string, the name of geometry column in the database table. (existing or to be created; defaults to 'geom').
create.table	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).
force.match	character, schema and table of the PostgreSQL table to compare columns of data frame with. If specified with partial.match = TRUE 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.
conn	A database connection (if a table is given in for "force.match" parameter)
new.id	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").

alter.names	Logical, whether to make database column names DB-compliant (remove special characters). Defualt is TRUE. (This should to be set to FALSE to match to non-standard names in an existing database table using the force.match setting.)
partial.match	Logical; if force.match is set and true, columns in R data frame will be compared with an the existing database table name. Only columns in the data frame that exactly match the database table will be inserted into the database table.
x	A list of class pgi, output from the pgInsertize() or pgInsertizeGeom() functions from the rpostgis package.
...	Further arguments not used.

Details

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 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 for some spatial types (faster with large datasets), otherwise the rgeos function writeWKT is used.

Value

pgi A list containing four character strings: (1) in.table, the table name which will be created or inserted into, if specifed 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 <dbucklin@ufl.edu>

Examples

```
## Not run:
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)

## 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 = conn, data.obj = 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 = conn, data.obj = pgi.existing)

## End(Not run)
## Not run:
## Format regular (non-spatial) data frame for insert using
## pgInsertize connect to database
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)

## 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, data.obj = 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, data.obj = values)

## End(Not run)
```

`pgListGeomTables` *List geometries.*

Description

List tables with geometry columns in the database.

Usage

```
pgListGeomTables(conn)
```

Arguments

conn	A PostgreSQL database connection

Value

A data frame with schema, table, geometry column, and geometry type.

Examples

```
## Not run:
pgListGeomTables(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 = "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

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

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

## Create a new LINESTRING field called "stp_geom"
pgMakeStp(conn, 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@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 = FALSE, new.srid = NULL)
```

Arguments

`conn`	A connection object to a PostgreSQL database.
`crs`	CRS object, created through a call to CRS.
`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 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 = 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", password = "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, *20*

dbAddKey, 2
dbAsDate, 3
dbColumn, 4
dbComment, 5
dbDrop, 6
dbIndex, 7
dbSchema, 8
dbTableInfo, 9
dbVacuum, 9

pgGet *(*pgGetPts*)*, 11
pgGetBoundary, 10
pgGetLines *(*pgGetPts*)*, 11
pgGetPolys *(*pgGetPts*)*, 11
pgGetPts, 11
pgGetRast, 12
pgInsert, 13
pgInsertize *(*pgInsertizeGeom*)*, 15
pgInsertizeGeom, 15
pgListGeomTables, 17
pgMakePts, 18
pgMakeStp *(*pgMakePts*)*, 18
pgPostGIS, 19
pgSRID, 20
print.pgi *(*pgInsertizeGeom*)*, 15

rpostgis, 21
rpostgis-package *(*rpostgis*)*, 21