pgUtils

March 24, 2012

`AutoReference-class`
Class "AutoReference", define references (relations) between database tables

---

Description

Objects of the type AutoReference are used in the functions `insertIntoTable` and `updateDBTable` to define relations between database tables.

Objects from the Class

Objects can be created by calls of the form `new("AutoReference", ...)`. Usually the object should be created with `new("AutoReference",source.table="<source table name>",ref.table="ref table name",source.table.column="<column in data>",ref.tab in ref table>")`

Slots

`source.table`: Object of class `"character"` the name of the source table (that will contain later the foreign key to the referenced table)

`ref.table`: Object of class `"character"` the name of the referenced table

`source.table.column`: Object of class `"character"` the column name of the data submitted thats value is the same as the value in the ref table, attribute ref.table.column

`ref.table.column`: Object of class `"character"` the column (attribute) name, where the value of the source.table.column should compared with

Methods

getRefAttribute `signature(object = "AutoReference")`: returns the primary key of the referenced table (<ref.table.name>_pk)

getRefColumn `signature(object = "AutoReference")`: returns the ref.table.column parameter

getRefTable `signature(object = "AutoReference")`: returns the name of the referenced table

getSourceAttribute `signature(object = "AutoReference")`: returns the name of the foreign key in the source table (usually <ref.table.name>_fk)

getSourceColumn `signature(object = "AutoReference")`: returns the source.table.column attribute

show `signature(object = "AutoReference")`: show method

Author(s)

Johannes Rainer

See Also

`createDBTable`, `insertIntoTable`, `updateDBTable`

`auto.fk.name` *Generating the foreign key name for a referenced table*

Description

`auto.fk.name` this function creates a foreign key name for the referenced table name submitted (simply by appending "_fk").

Usage

```
auto.fk.name(name)
```

Arguments

`name`	The table name.

Author(s)

Johannes Rainer

`auto.pk.name` *Generating the primary key name for a table*

Description

`auto.pk.name` this function creates a primary key name for the table name submitted (simply by appending "_pk").

Usage

```
auto.pk.name(name)
```

Arguments

`name`	The table name.

Author(s)

Johannes Rainer

auto.pk.seq — *Generating the name for the SEQUENCE for a primary key*

Description

`auto.pk.seq` this function creates the name of the SEQUENCE to be created for a auto incrementing primary key.

Usage

```
auto.pk.seq(name)
```

Arguments

`name`	The table name.

Author(s)

Johannes Rainer

createDBTable — *Creating a PgSQL database table*

Description

`createDBTable` creates a PostgreSQL data base table with primary key and possible foreign keys if needed (to guarantee the referencial integrity).

Usage

```
createDBTable(conn, name=NULL, attributes=NULL, data.types=NULL, options=NULL, r
```

Arguments

`conn`	A connection object (create it with the dbConnect.PgSQL.conn function from the package RdbiPgSQL)
`name`	The table name
`attributes`	a vector containing the attribute names for the table
`data.types`	a vector specifying the data type for each attribute. If not submitted every column attribute is set to the TEXT data type. The data types can be defined by submitting a vector with the SQL data types for the attributes. Valid data types are: TEXT for character, BOOL for logical, REAL for numeric (real, for integer also INTEGER can be submitted). If the data types are all the same for all attributes, only one data type can be submitted (with the SQL data types above or, in the case it is not(!) a character an exemplary value can be submitted (e.g. `data.types=2`, so the data types for all attributes in the database table are set to REAL)).
`options`	optional, some advanced options for the attributes. If submitted it must have the same length as the attributes argument.

`references` the table name(s) where the foreign key(s) should point to.

`references.options`
a vector with the same length of references, that allows to specify some options for the refernces (for example if one wants to allow foreign keys to get a NULL value, for example if no row in the references table exists where this row can reference to as references.options a "" can be submitted. If nothing is submitted (the default case), for every reference (foreign key) the constraint NOT NULL will be inserted, so no NULL references are allowed.)

`no.clobber` if already existant tables with the same name should be deleted, default is FALSE

`auto.pk` if a primary key for this table should be create automatically (default is TRUE, so a autoincrementing primary key will be created for that table. Everytime a new row is inserted into the table, the primary key will be created automatically).

`no.transaction`
if the whole call should be performed into a TRANSACTION, so if any error occurrs during the processing of the function, the database will not be touched. The default value is TRUE, so the function is quite robust.

`silent` if the function should not check wether or not the table does already exist.

Details

creates a PostgreSQL database table, with the possibility to generate a primary key column that is incremented automatically with each insertion of data. Also foreign keys, that are references to other tables will be generated. So the referential integrity of the data in the tables is warranted. Advanced options can be specified for the tables attributes too (such as the keyword UNIQUE for a column).

Author(s)

Johannes Rainer

See Also

`insertIntoTable`, `updateDBTable`

Examples

```
## create a simple table called "individual", that references to a already existant tab
## Not run: Con <- dbConnect(PgSQL(),user="postgres",dbname="template1",host="localhost
## Not run: createDBTable(Con,name="individual",attributes=c("name","age"),data.types=c

## this call creates the database table "individual", with the primary key "individual_
## "name" and "age" and the foreign key "species_fk", that will, upon inserting of any
## primary key of the table "species".
```

`createSequence` *Creating a PgSQL SEQUENCE*

Description

`createSequence` creates a PostgreSQL SEQUENCE. SEQUENCEs allow the creation of autoincrementing sequences.

Usage

```
createSequence(con, name=NULL, increment=1, start=100)
```

Arguments

con	A connection object (create it with the dbConnect.PgSQL.conn function from the package RdbiPgSQL)
name	The name for the sequence
increment	the value to increment the sequence by each call, by default 1.
start	the starting value of the sequence, by default 100.

Details

creates a PostgreSQL SEQUENCE table. Such a sequence table can be used to get unique numbers and are therefore ideal for Primary keys (by creating a table with the default call createDBTable also a sequence will be created, to get unique Primary keys for the rows). To get the next value from the sequece only call nextval('<sequence name>'). For more details look in the PostgreSQL reference manual.

Author(s)

Johannes Rainer

See Also

createDBTable

---

dbColnames *Get the column names of a database table*

Description

This function returns the column names of the database table which name was submitted.

Usage

```
dbColnames(con, table.name)
```

Arguments

con	The database connection object.
table.name	The database table name for which the column names should be retrieved.

Value

The column names of the table.

Author(s)

Johannes Rainer

dbListDatabases *List all databases*

Description

This function lists all databases from an PostgreSQL backend.

Usage

```
dbListDatabases(con)
```

Arguments

con	The database connection object.

Value

A data.frame with the database names and all other properties of the databases

Author(s)

Johannes Rainer

dbListSequences *Listing all SEQUENCES in a PostgreSQL database.*

Description

`dbListSequences` gets the names of all sequences in the database.

Usage

```
dbListSequences(conn, pattern, all)
```

Arguments

conn	A connection object (create it with the dbConnect.PgSQL.conn function from the package RdbiPgSQL)
pattern	Search for sequences that contain the pattern submitted in their name
all	If TRUE also system sequences will be listed. The default is FALSE.

Details

Returns the names of all sequences in the database to wich the user has connected.

Author(s)

Johannes Rainer

See Also

`createSequence`

`deleteDBEntry` *Deleting a row (or all rows) in a database table*

Description

`deleteDBEntry` With this function a row or all rows can be deleted from a database table. If the rows to be deleted are referenced by another table the corresponding row will not be deleted (to keep the referencial integrity).

Usage

```
deleteDBEntry(con,name=NULL,attribute=NULL,value=NULL,condition="=", no.transact
```

Arguments

`con`	A connection object (create it with the dbConnect.PgSQL.conn function from the package RdbiPgSQL)
`name`	The table name
`attribute`	The column name in the database where the values in the `values` argument can be found (the corresponding rows will then be deleted).
`value`	The values that should be compared with the entries in the column of the database table (specified by the `attribute` argument). If nothing is submitted, all rows in the database will be deleted.
`condition`	The condition that has to be fulfilled by the values (submitted with the `value` argument and the values in the database table, specified by the `attribute` column).
`no.transaction`	if the whole call should not be performed into a TRANSACTION, so if any error occurrs during the processing of the function, the database will not be touched. The default value is FALSE, so the function is quite robust.

Details

For the people that know already a little bit of SQL, this function does nothing else as sending the query DELETE FROM <name> WHERE <attribute><condition><value>;.

Author(s)

Johannes Rainer

See Also

`createDBTable`

do.log *If SQL command output should be logged*

Description

do.log

Usage

```
do.log
```

Details

Set this variable to FALSE if nothing should be logged. By default do.log is TRUE

Author(s)

Johannes Rainer

See Also

log.file log.level loggit

getRefAttribute-methods *Return the name of the primary key of the referenced table*

Description

getRefAttribute returns the primary key attribute name of the referenced table (usually the table name followed by "_pk")

Methods

object = "AutoReference" The AutoReference object.

See Also

AutoReference-class

getRefColumn-methods

Get the referenced column name

Description

`getRefColumn`

Methods

object = "AutoReference" The AutoReference-class object.

See Also

`AutoReference-class`

getRefTable-methods

Get the name of the referenced table

Description

`getRefTable` in package **tkfiDB** returns the name of the referenced table.

Methods

object = "AutoReference" An instance of the object AutoReference-class

See Also

`AutoReference-class`

getSourceAttribute-methods

Get the source attribute

Description

`getSourceAttribute` in package **tkfiDB** returns the attribute of the source database table that should be used to establish the connection to the referenced database table.

Methods

object = "AutoReference" An instance of the object AutoReference-class

See Also

`AutoReference-class`

getSourceColumn-methods

Get the column name of the source table

Description

`getSourceColumn` in package **tkfiDB** returns the name of the source column.

Methods

object = "AutoReference" An instance of the class AutoReference-class

See Also

`AutoReference-class`

insertIntoTable

Adding new data to a database table

Description

`insertIntoTable` With this function new data rows can be inserted into an existant database table. Automatically generated Primary keys will be updated (more in detail, if the table was created with the `createDBTable` function and the argument `auto.pk=TRUE`, in the primary key attribute in the database table a new (unique) number will be inserted (using a PostgreSQL specific SEQUENCE table)). Foreign keys, that are references to other tables, will also be inserted if needed. The referencial integrity will always be guaranteed, as the function allows no insertion of data with a reference to another entry in another table, that does not exist.

Usage

```
insertIntoTable(con, name=NULL, data=NULL, attributes=NULL, references=NULL, no.
```

Arguments

`con`	A connection object (create it with the dbConnect.PgSQL.conn function from the package RdbiPgSQL)
`name`	The table name.
`data`	The data that should be inserted, in form of a single value, a vector, matrix, data.frame...
`attributes`	a vector containing the attribute names for the table. This parameter can be omitted, if as data argument a matrix or data.frame with the corresponding column names is submitted.
`references`	a list, or a single object of the type AutoReference, that represents the references to other tables, so that foreign keys can be updated/generated. If the entry should reference to more then one table, create a `list` of your `AutoReference` objects and submit this list.
`no.transaction`	if the whole call should be performed into a TRANSACTION, so if any error occurrs during the processing of the function, the database will not be touched. The default value is FALSE, so the function is quite robust.

Details

This function allows a simple filling of a database table with a single value or a whole table, as well as some more sophisticated operations, like the auto incrementing of a primary key (this will be automatically done if the table was created with the `createDBTable` function)). Another important feature is the possibility to resolve references to guarantee for a referential integrity. To use this option submit one or more `AutoReference` object(s) with the `references` argument. Remember that you have to insert on or more (depending of the number of references) columns in your data table, in which you have to define the values from the referenced table. From the corresponding rows in the referenced table the primary key will then be taken as foreign key in the table where the values are gone to be written into. More in detail the row to be inserted will be linked to the row in the referenced table, where the value in the <ref.table.column> attribute of the `AutoReference` object is the same as the value in the according column of the <data> object, that is specified in the <source.table.column> attribute of the `AutoReference` object.

Author(s)

Johannes Rainer

See Also

`AutoReference-class`, `createDBTable`

Examples

```
## creating a connection
## Not run: Con <- dbConnect(PgSQL(),user="postgres",host="localhost",dbname="template1")

## creating a table with two attributes name and value. as third attribute
## a primary key with the name atable_pk will automatically generated.
## Not run: createDBTable(Con,name="atable",attributes=c("name","value"),data.types=c("TE

## fill some data into the table, remember that not all cells have to be
## filled.
## Not run: insertIntoTable(Con,name="atable",data=data.frame(species=c("hobbit","human")

## now create a table that references this table.
## Not run: createDBTable(Con,name="btable",attributes=c("name","age"),data.types=c("TEXT

## before inserting any values to this table we have to create a reference.
## the source.table.column specifies the column in the submitted data, which
## values are equal to the values in the ref.table.column attribute in the
## referenced table.
## Not run: ref <- new("AutoReference",source.table="btable",ref.table="atable",source.ta

## Not run: insertIntoTable(Con,name="btable",data=data.frame(name=c("frodo","fred"),aref

## summary:
## the function inserts in the table btable "frodo" in the attribute <name>, and
## links this row to the referenced table btable, by setting in the attribute
## <atable_fk> (in table <btable>) the value from the primary key of the row in
## the table <atable>, where the attribute <species> has the value "hobbit".

## if we had submitted in the aref column a value that is not present in the
## atable, a error would be thrown and nothing is going to be written into
## the table. So the referential integrity is guaranteed. Naturally we can insert into th
```

`log.file` *The log file for the SQL commands*

Description

`log.file`.

Usage

```
log.file
```

Details

With this variable the log file can be defined. By default all output will be logged into a file "tkfi.log" in the current working directory.

Author(s)

Johannes Rainer

See Also

`do.log` `log.level` `loggit`

`log.level` *Log level*

Description

The `log.level` variable can be used to set the system log level. Valid values are "ERROR" and "DEBUG". In the first case only errors will be logged, in the second everithing message will be logged to the log file.

Usage

```
log.level
```

Author(s)

Johannes Rainer

See Also

`log.file` `do.log` `loggit`

loggit *Logging errors or debuggin information*

Description

`loggit` this function writes the desired information into a log file. The log file can be specifyed with the system variable "log.file" (simply write `log.file <- "YourLogFile.txt"` in your R console). When the system variable "log.level" is "DEBUG" everytime the function is called the information is written to the log file, if the logging level was set to "ERROR" only those calls are written into the log file, that produced a SQL error. In all functions of this package the loggit function is only called if the system variable `do.log` is set to TRUE. Together with the information submitted, the date and time of the call will be written to the log file.

Usage

```
loggit(resultSet,call="")
```

Arguments

`resultSet`	The result set that is returned by every query that is sent to the database (eg with the `dbSendQuery` command).
`call`	Some optional description that should be appended.

Author(s)

Johannes Rainer

searchIn *Perform a pattern search in a database table*

Description

`searchIn` With this function it is possible to search a word or a pattern string in a database table. The function returns all the rows from the table where the string was found.

Usage

```
searchIn(Con,name=NULL,pattern=NULL,where=NULL)
```

Arguments

`Con`	A connection object (create it with the dbConnect.PgSQL.conn function from the package RdbiPgSQL)
`name`	The table name
`pattern`	The string or part of the string that should be searched.
`where`	An optional parameter that specifies the column where the string should be searched in the table. If not submitted, the function searches the string in all columns that are from the SQL data type TEXT.

Details

This function searches the string submitted with the SQL command `LIKE %pattern%`. So the string to search can be inside a word, or be the word itself. As a pattern search is only defined for text strings, the function will search only in those columns that contain text. So for all columns that have some SQL datatype different from TEXT (for example BOOL or INT) no search is performed.

Author(s)

Johannes Rainer

`updateDBTable` *Updating information in a database table*

Description

`updateDBTable` this function is, as the name already says, a wrapper method for the SQL UPDATE command. So information in a database table can be updated with this function, or missing data in a table row can be inserted. Also references can be updated.

Usage

```
updateDBTable(con=NULL, name=NULL, set.col=NULL, where.col=NULL, new.values=NULL
```

Arguments

`con`	A connection object (create it with the dbConnect.PgSQL.conn function from the package RdbiPgSQL)
`name`	The table name
`set.col`	The column(s) that should be updated in the table. This argument can be omitted, if it is the same as the column names in a matrix or data.frame submitted as new.values.
`where.col`	The column(s) that can be used to search for the rows to update. This argument can also be omitted, if it is the same as the column names in the data.frame or matrix submitted as old.values argument.
`new.values`	The new value(s) that should be inserted in the table (or with wich old values should be updated).
`old.values`	The old values on which the function can perform a query to specify the rows that should be updated. In the case a references attribute is submitted, this matrix (or vector, or matrix) should contain a column, that contains values that are the same as those in the referenced table, to wich the specific row in the table to be updated should be linked (referenced) to.
`condition`	With this argument, the type of comparison can be specified, the default value is "=", but all other regular SQL queries can be used. To find the row to be updated the question <where.cols><condition><old.values> will be asked, so in the default case: <where.cols>=<old.values> is asked.

`references` a list, or a single object of the type AutoReference, that represents the references to other tables, so that foreign keys can be updated/generated. Such a AutoReference object has to contain the name of the table to be updated, the referenced table, the name of the column in the <old.values> argument and the column name in the referenced table. So the row to be inserted will be linked to the row in the referenced table, where the value in the <ref.table.column> attribute of the `AutoReference` object is the same as the value in the according column, that is specified in the <source.table.column> attribute of the `AutoReference` object, of the <old.values> argument.

`no.transaction` if the whole call should not be performed into a TRANSACTION, so if any error occurrs during the processing of the function, the database will not be touched. The default value is FALSE, so the function is quite robust.

Details

For the people that know already a little bit of SQL, this function does nothing else as sending the query UPDATE <name> SET <set.cols>=<new.values> WHERE <where.cols><conditon><old.values>. It does this for a data.frame, matrix or single values submitted. So a whole table can be updated, some of the fields in a table or only one value. With the `reference` attribute also references to other tables can be updated. To do this, one or more columns (depending on the number of references to other tables) have to be inserted in the <old.values> attribute, so that the function can update the reference upon the value in this column.

Author(s)

Johannes Rainer

See Also

`AutoReference-class`, `createDBTable`

Index