

Package ‘DBI’

September 24, 2014

Version 0.3.1

Title R Database Interface

Author R Special Interest Group on Databases (R-SIG-DB)

Maintainer Hadley Wickham <hadley@rstudio.com>

Depends R (>= 2.15.0), methods

Suggests testthat, RSQLite

Description A database interface (DBI) definition for communication between R and relational database management systems. All classes in this package are virtual and need to be extended by the various R/DBMS implementations.

License LGPL (>= 2)

URL <https://github.com/rstats-db/DBI>

BugReports <https://github.com/rstats-db/DBI/issues>

Collate 'DBObject.R' 'DBConnection.R' 'DBDriver.R' 'DBResult.R' 'compliance.R' 'keywords.R' 'quote.R' 'util.R'

NeedsCompilation no

Repository CRAN

Date/Publication 2014-09-24 07:27:12

R topics documented:

dbClearResult	2
dbColumnInfo	3
dbConnect	4
dbDataType	5
dbDisconnect	6
dbDriver	6

dbExistsTable	7
dbFetch	8
dbGetException	9
dbGetInfo	10
dbGetQuery	11
dbGetRowCount	12
dbGetRowsAffected	12
dbGetStatement	13
dbHasCompleted	13
dbiCheckCompliance	14
DBIConnection-class	14
DBIDriver-class	15
DBIObject-class	15
DBIResult-class	16
dbIsValid	17
dbListConnections	17
dbListFields	18
dbListResults	18
dbListTables	19
dbReadTable	19
dbRemoveTable	20
dbSendQuery	21
make.db.names	22
SQL	23
transactions	25

Index 26

dbClearResult	<i>Clear a result set.</i>
---------------	----------------------------

Description

Frees all resources (local and remote) associated with a result set. In some cases (e.g., very large result sets) this can be a critical step to avoid exhausting resources (memory, file descriptors, etc.)

Usage

```
dbClearResult(res, ...)
```

Arguments

res	An object inheriting from DBIResult .
...	Other arguments passed on to methods.

Value

a logical indicating whether clearing the result set was successful or not.

See Also

Other DBIResult generics: [dbColumnInfo](#); [dbFetch](#), [dbFetch](#), [DBIResult-method](#), [fetch](#); [dbGetRowCount](#), [dbGetRowCount](#), [DBIResult-method](#); [dbGetRowsAffected](#), [dbGetRowsAffected](#), [DBIResult-method](#); [dbGetStatement](#), [dbGetStatement](#), [DBIResult-method](#); [dbHasCompleted](#), [dbHasCompleted](#), [DBIResult-method](#)

dbColumnInfo	<i>Information about result types.</i>
--------------	--

Description

Produces a data.frame that describes the output of a query. The data.frame should have as many rows as there are output fields in the result set, and each column in the data.frame should describe an aspect of the result set field (field name, type, etc.)

Usage

```
dbColumnInfo(res, ...)
```

Arguments

res	An object inheriting from DBIResult .
...	Other arguments passed on to methods.

Value

A data.frame with one row per output field in res. Methods MUST include name, field.type (the SQL type), and data.type (the R data type) columns, and MAY contain other database specific information like scale and precision or whether the field can store NULLs.

See Also

Other DBIResult generics: [dbClearResult](#); [dbFetch](#), [dbFetch](#), [DBIResult-method](#), [fetch](#); [dbGetRowCount](#), [dbGetRowCount](#), [DBIResult-method](#); [dbGetRowsAffected](#), [dbGetRowsAffected](#), [DBIResult-method](#); [dbGetStatement](#), [dbGetStatement](#), [DBIResult-method](#); [dbHasCompleted](#), [dbHasCompleted](#), [DBIResult-method](#)

`dbConnect`*Create a connection to a DBMS.*

Description

Connect to a DBMS going through the appropriate authorization procedure. Some implementations may allow you to have multiple connections open, so you may invoke this function repeatedly assigning its output to different objects. The authorization mechanism is left unspecified, so check the documentation of individual drivers for details.

Usage

```
dbConnect(drv, ...)
```

Arguments

<code>drv</code>	an object that inherits from DBIDriver , or a character string specifying the name of DBMS driver, e.g., "SQLite", "RMySQL", "RPostgreSQL", or an existing DBIConnection object (in order to clone an existing connection).
<code>...</code>	authorization arguments needed by the DBMS instance; these typically include user, password, dbname, host, port, etc. For details see the appropriate DBIDriver .

Details

Each driver will define what other arguments are required, e.g., "dbname" for the database name, "username", and "password".

Value

An object that extends [DBIConnection](#) in a database-specific manner. For instance `dbConnect("MySQL")` produces an object of class `MySQLConnection`. This object is used to direct commands to the database engine.

See Also

[dbDisconnect](#) to disconnect from a database.

Examples

```
if (require("SQLite")) {
# SQLite only needs a path to the database. Other database drivers
# will require more details (like username, password, host, port etc)
con <- dbConnect(RSQLite::SQLite(), ":memory:")
con

dbListTables(con)
dbDisconnect(con)
}
```

dbDataType	<i>Determine the SQL data type of an object.</i>
------------	--

Description

This is a generic function. The default method determines the SQL type of an R object according to the SQL 92 specification, which may serve as a starting point for driver implementations. The default method also provides a method for `data.frame` which will return a character vector giving the type for each column in the dataframe.

Usage

```
dbDataType(dbObj, obj, ...)
```

Arguments

dbObj	A object inheriting from <code>DBIDriver</code>
obj	An R object whose SQL type we want to determine.
...	Other arguments passed on to methods.

Details

The data types supported by databases are different than the data types in R, but the mapping between the primitive types is straightforward: Any of the many fixed and varying length character types are mapped to character vectors. Fixed-precision (non-IEEE) numbers are mapped into either numeric or integer vectors.

Notice that many DBMS do not follow IEEE arithmetic, so there are potential problems with under/overflows and loss of precision.

Value

A character string specifying the SQL data type for `obj`.

See Also

[isSQLKeyword](#) [make.db.names](#)

Examples

```
if (require("RSQLite")) {
  con <- dbConnect(RSQLite::SQLite(), ":memory:")

  dbDataType(con, 1:5)
  dbDataType(con, 1L)
  dbDataType(con, TRUE)
  dbDataType(con, c("x", "abc"))

  dbDataType(con, mtcars)
}
```

dbDisconnect	<i>Disconnect (close) a connection</i>
--------------	--

Description

This closes the connection, discards all pending work, and frees resources (e.g., memory, sockets).

Usage

```
dbDisconnect(conn, ...)
```

Arguments

conn	A DBIConnection object, as produced by dbConnect .
...	Other parameters passed on to methods.

Value

a logical vector of length 1, indicating success or failure.

See Also

Other connection methods: [dbExistsTable](#); [dbGetException](#); [dbGetQuery](#), [dbGetQuery](#), [DBIConnection](#), [character-me](#); [dbListFields](#); [dbListResults](#); [dbListTables](#); [dbReadTable](#), [dbWriteTable](#); [dbRemoveTable](#); [dbSendQuery](#)

Examples

```
if (require("RSQLite")) {
  con <- dbConnect(RSQLite::SQLite(), ":memory:")
  dbDisconnect(con)
}
```

dbDriver	<i>Load and unload database drivers.</i>
----------	--

Description

dbDriver is a helper method used to create a new driver object given the name of a database or the corresponding R package. It works through convention: all DBI-extending packages should provide an exported object with the same name as the package. dbDriver just looks for this object in the right places: if you know what database you are connecting to, you should call the function directly.

Usage

```
dbDriver(drvName, ...)
```

```
dbUnloadDriver(drv, ...)
```

Arguments

drvName character name of the driver to instantiate.
 ... any other arguments are passed to the driver drvName.
 drv an object that inherits from DBIDriver as created by dbDriver.

Value

In the case of dbDriver, an driver object whose class extends DBIDriver. This object may be used to create connections to the actual DBMS engine.

In the case of dbUnloadDriver, a logical indicating whether the operation succeeded or not.

Side Effects

The client part of the database communication is initialized (typically dynamically loading C code, etc.) but note that connecting to the database engine itself needs to be done through calls to dbConnect.

Examples

```
if (require("RSQLite")) {
# Create a RSQLite driver with a string
d <- dbDriver("SQLite")
d

# But better, access the object directly
RSQLite::SQLite()
}
```

dbExistsTable

Does a table exist?

Description

Does a table exist?

Usage

```
dbExistsTable(conn, name, ...)
```

Arguments

conn	A DBIConnection object, as produced by dbConnect .
name	A character string specifying a DBMS table name.
...	Other parameters passed on to methods.

Value

a logical vector of length 1.

See Also

Other connection methods: [dbDisconnect](#); [dbGetException](#); [dbGetQuery](#), [dbGetQuery](#), [DBIConnection](#), [character-met](#), [dbListFields](#); [dbListResults](#); [dbListTables](#); [dbReadTable](#), [dbWriteTable](#); [dbRemoveTable](#); [dbSendQuery](#)

 dbFetch

Fetch records from a previously executed query.

Description

Fetch the next *n* elements (rows) from the result set and return them as a `data.frame`.

Usage

```
dbFetch(res, n = -1, ...)
```

```
fetch(res, n = -1, ...)
```

Arguments

res	An object inheriting from DBIResult .
n	maximum number of records to retrieve per fetch. Use <code>n = -1</code> to retrieve all pending records. Some implementations may recognize other special values.
...	Other arguments passed on to methods.

Details

`fetch` is provided for compatibility with older DBI clients - for all new code you are strongly encouraged to use `dbFetch`. The default method for `dbFetch` calls `fetch` so that it is compatible with existing code. Implementors should provide methods for both `fetch` and `dbFetch` until `fetch` is deprecated in 2015.

Value

a `data.frame` with as many rows as records were fetched and as many columns as fields in the result set.

See Also

close the result set with `dbClearResult` as soon as you finish retrieving the records you want.

Other DBIResult generics: `dbClearResult`; `dbColumnInfo`; `dbGetRowCount`, `dbGetRowCount`, `DBIResult-method`; `dbGetRowsAffected`, `dbGetRowsAffected`, `DBIResult-method`; `dbGetStatement`, `dbGetStatement`, `DBIResult-method`; `dbHasCompleted`, `dbHasCompleted`, `DBIResult-method`

Examples

```
if (!require("RSQLite")) {
  con <- dbConnect(RSQLite::SQLite(), ":memory:")
  dbWriteTable(con, "mtcars", mtcars)

  # Fetch all results
  res <- dbSendQuery(con, "SELECT * FROM mtcars WHERE cyl = 4")
  dbFetch(res)
  dbClearResult(res)

  # Fetch in chunks
  res <- dbSendQuery(con, "SELECT * FROM mtcars")
  while (!dbHasCompleted(res)) {
    chunk <- fetch(res, 10)
    print(nrow(chunk))
  }
  dbClearResult(res)
  dbDisconnect(con)
}
```

dbGetException	<i>Get DBMS exceptions.</i>
----------------	-----------------------------

Description

Get DBMS exceptions.

Usage

```
dbGetException(conn, ...)
```

Arguments

conn	A <code>DBIConnection</code> object, as produced by <code>dbConnect</code> .
...	Other parameters passed on to methods.

Value

a list with elements `errNum` (an integer error number) and `errMsg` (a character string) describing the last error in the connection `conn`.

See Also

Other connection methods: [dbDisconnect](#); [dbExistsTable](#); [dbGetQuery](#), [dbGetQuery](#), [DBIConnection](#), [character-method](#), [dbListFields](#); [dbListResults](#); [dbListTables](#); [dbReadTable](#), [dbWriteTable](#); [dbRemoveTable](#); [dbSendQuery](#)

dbGetInfo	<i>Get DBMS metadata.</i>
-----------	---------------------------

Description

Get DBMS metadata.

Usage

```
dbGetInfo(dbObj, ...)
```

Arguments

dbObj	An object inheriting from DBIObject , i.e. DBIDriver , DBIConnection , or a DBIResult
...	Other arguments to methods.

Value

a named list

Implementation notes

For [DBIDriver](#) subclasses, this should include the version of the package (`driver.version`), the version of the underlying client library (`client.version`), and the maximum number of connections (`max.connections`).

For [DBIConnection](#) objects this should report the version of the DBMS engine (`db.version`), database name (`dbname`), username (`username`), host (`host`), port (`port`), etc. It MAY also include any other arguments related to the connection (e.g., thread id, socket or TCP connection type). It MUST NOT include the password.

For [DBIResult](#) objects, this should include the statement being executed (`statement`), how many rows have been fetched so far (in the case of queries) (`row.count`), how many rows were affected (deleted, inserted, changed, or total number of records to be fetched). (`rows.affected`), if the query is complete (`has.completed`), and whether or not the query generates output (`is.select`).

See Also

Other [DBObject](#) methods: [dbIsValid](#)

dbGetQuery	<i>Send query, retrieve results and then clear result set.</i>
------------	--

Description

dbGetQuery comes with a default implementation that calls [dbSendQuery](#), then if [dbHasCompleted](#) is TRUE, it uses [fetch](#) to return the results. [on.exit](#) is used to ensure the result set is always freed by [dbClearResult](#). Subclasses should override this method only if they provide some sort of performance optimisation.

Usage

```
dbGetQuery(conn, statement, ...)
```

Arguments

conn	A DBIConnection object, as produced by dbConnect .
statement	a character vector of length 1 containing SQL.
...	Other parameters passed on to methods.

See Also

Other connection methods: [dbDisconnect](#); [dbExistsTable](#); [dbGetException](#); [dbListFields](#); [dbListResults](#); [dbListTables](#); [dbReadTable](#), [dbWriteTable](#); [dbRemoveTable](#); [dbSendQuery](#)

Examples

```
if (require("RSQLite")) {  
  con <- dbConnect(RSQLite::SQLite(), ":memory:")  
  
  dbWriteTable(con, "mtcars", mtcars)  
  res <- dbSendQuery(con, "SELECT * FROM mtcars WHERE cyl = 4;")  
  dbFetch(res)  
  dbClearResult(res)  
  
  dbDisconnect(con)  
}
```

dbGetRowCount	<i>The number of rows fetched so far.</i>
---------------	---

Description

The default method extracts `row.count` from the result of `dbGetInfo(res)`.

Usage

```
dbGetRowCount(res, ...)
```

Arguments

<code>res</code>	An object inheriting from <code>DBIResult</code> .
<code>...</code>	Other arguments passed on to methods.

Value

a numeric vector of length 1

See Also

Other `DBIResult` generics: `dbClearResult`; `dbColumnInfo`; `dbFetch`, `dbFetch`, `DBIResult-method`, `fetch`; `dbGetRowsAffected`, `dbGetRowsAffected`, `DBIResult-method`; `dbGetStatement`, `dbGetStatement`, `DBIResult-method`; `dbHasCompleted`, `dbHasCompleted`, `DBIResult-method`

dbGetRowsAffected	<i>The number of rows affected by data modifying query.</i>
-------------------	---

Description

The default method extracts `rows.affected` from the result of `dbGetInfo(res)`.

Usage

```
dbGetRowsAffected(res, ...)
```

Arguments

<code>res</code>	An object inheriting from <code>DBIResult</code> .
<code>...</code>	Other arguments passed on to methods.

Value

a numeric vector of length 1

See Also

Other DBIResult generics: [dbClearResult](#); [dbColumnInfo](#); [dbFetch](#), [dbFetch](#), [DBIResult-method](#), [fetch](#); [dbGetRowCount](#), [dbGetRowCount](#), [DBIResult-method](#); [dbGetStatement](#), [dbGetStatement](#), [DBIResult-method](#); [dbHasCompleted](#), [dbHasCompleted](#), [DBIResult-method](#)

dbGetStatement	<i>Get the statement associated with a result set</i>
----------------	---

Description

The default method extracts statement from the result of [dbGetInfo](#)(res).

Usage

```
dbGetStatement(res, ...)
```

Arguments

res	An object inheriting from DBIResult .
...	Other arguments passed on to methods.

Value

a character vector

See Also

Other DBIResult generics: [dbClearResult](#); [dbColumnInfo](#); [dbFetch](#), [dbFetch](#), [DBIResult-method](#), [fetch](#); [dbGetRowCount](#), [dbGetRowCount](#), [DBIResult-method](#); [dbGetRowsAffected](#), [dbGetRowsAffected](#), [DBIResult-method](#); [dbHasCompleted](#), [dbHasCompleted](#), [DBIResult-method](#)

dbHasCompleted	<i>Has the operation completed?</i>
----------------	-------------------------------------

Description

The default method extracts has.completed from the result of [dbGetInfo](#)(res).

Usage

```
dbHasCompleted(res, ...)
```

Arguments

res	An object inheriting from DBIResult .
...	Other arguments passed on to methods.

Value

a logical vector of length 1

See Also

Other DBIResult generics: [dbClearResult](#); [dbColumnInfo](#); [dbFetch](#), [dbFetch](#), [DBIResult-method](#), [fetch](#); [dbGetRowCount](#), [dbGetRowCount](#), [DBIResult-method](#); [dbGetRowsAffected](#), [dbGetRowsAffected](#), [DBIResult-me](#); [dbGetStatement](#), [dbGetStatement](#), [DBIResult-method](#)

<code>dbiCheckCompliance</code>	<i>Check a driver for compliance with DBI.</i>
---------------------------------	--

Description

Check a driver for compliance with DBI.

Usage

```
dbiCheckCompliance(driver, pkg = paste0("R", driver))
```

Arguments

<code>driver</code>	Driver name.
<code>pkg</code>	Package that driver lives in - is usually "Rdriver"

Examples

```
if (require("RSQLite")) {
  dbiCheckCompliance("SQLite")
  dbiCheckCompliance("NoDriver", "RSQLite")
}
```

<code>DBIConnection-class</code>	<i>DBIConnection class.</i>
----------------------------------	-----------------------------

Description

This virtual class encapsulates the connection to a DBMS, and it provides access to dynamic queries, result sets, DBMS session management (transactions), etc.

Implementation note

Individual drivers are free to implement single or multiple simultaneous connections.

See Also

Other DBI classes: [DBIDriver-class](#); [DBIObject-class](#); [DBIResult-class](#)

Examples

```
## Not run:
con <- dbConnect(RSQLite::SQLite(), ":memory:")
dbDisconnect(con)

con <- dbConnect(RPostgreSQL::PostgreSQL(), "username", "password")
dbDisconnect(con)

## End(Not run)
```

DBIDriver-class	<i>DBIDriver class.</i>
-----------------	-------------------------

Description

Base class for all DBMS drivers (e.g., RSQLite, MySQL, PostgreSQL). The virtual class `DBIDriver` defines the operations for creating connections and defining data type mappings. Actual driver classes, for instance `RPgSQL`, `RMySQL`, etc. implement these operations in a DBMS-specific manner.

See Also

Other DBI classes: [DBIConnection-class](#); [DBIObject-class](#); [DBIResult-class](#)

DBIObject-class	<i>DBIObject class.</i>
-----------------	-------------------------

Description

Base class for all other DBI classes (e.g., drivers, connections). This is a virtual Class: No objects may be created from it.

Details

More generally, the DBI defines a very small set of classes and methods that allows users and applications access DBMS with a common interface. The virtual classes are `DBIDriver` that individual drivers extend, `DBIConnection` that represent instances of DBMS connections, and `DBIResult` that represent the result of a DBMS statement. These three classes extend the basic class of `DBIObject`, which serves as the root or parent of the class hierarchy.

Implementation notes

An implementation **MUST** provide methods for the following generics:

- [dbGetInfo](#).

It **MAY** also provide methods for:

- [summary](#). Print a concise description of the object. The default method invokes `dbGetInfo(dbObj)` and prints the name-value pairs one per line. Individual implementations may tailor this appropriately.

See Also

Other DBI classes: [DBIConnection-class](#); [DBIDriver-class](#); [DBIResult-class](#)

Examples

```
## Not run:
drv <- dbDriver("MySQL")
con <- dbConnect(drv, group = "rs-dbi")
res <- dbSendQuery(con, "select * from vitalSuite")
is(drv, "DBIObject") ## True
is(con, "DBIObject") ## True
is(res, "DBIObject")

## End(Not run)
```

DBIResult-class

DBIResult class.

Description

This virtual class describes the result and state of execution of a DBMS statement (any statement, query or non-query). The result set `res` keeps track of whether the statement produces output how many rows were affected by the operation, how many rows have been fetched (if statement is a query), whether there are more rows to fetch, etc.

Details

Individual drivers are free to allow single or multiple active results per connection.

See Also

Other DBI classes: [DBIConnection-class](#); [DBIDriver-class](#); [DBIObject-class](#)

dbIsValid	<i>Is this DBMS object still valid?</i>
-----------	---

Description

This generic tests whether a database object is still valid (i.e. it hasn't been disconnected or cleared).

Usage

```
dbIsValid(dbObj, ...)
```

Arguments

dbObj	An object inheriting from DBIObject , i.e. DBIDriver , DBIConnection , or a DBIResult
...	Other arguments to methods.

Value

a logical of length 1

See Also

Other DBObject methods: [dbGetInfo](#)

dbListConnections	<i>List currently open connections.</i>
-------------------	---

Description

Drivers that implement only a single connections **MUST** return a list containing a single element. If no connection are open, methods **MUST** return an empty list.

Usage

```
dbListConnections(drv, ...)
```

Arguments

drv	A object inheriting from DBIDriver
...	Other arguments passed on to methods.

Value

a list

dbListFields	<i>List field names of a remote table.</i>
--------------	--

Description

List field names of a remote table.

Usage

```
dbListFields(conn, name, ...)
```

Arguments

conn	A DBIConnection object, as produced by dbConnect .
name	a character string with the name of the remote table.
...	Other parameters passed on to methods.

Value

a character vector

See Also

[dbColumnInfo](#) to get the type of the fields.

Other connection methods: [dbDisconnect](#); [dbExistsTable](#); [dbGetException](#); [dbGetQuery](#), [dbGetQuery](#), [DBIConnection](#), [dbListResults](#); [dbListTables](#); [dbReadTable](#), [dbWriteTable](#); [dbRemoveTable](#); [dbSendQuery](#)

dbListResults	<i>A list of all pending results.</i>
---------------	---------------------------------------

Description

List of [DBIResult](#) objects currently active on the connection.

Usage

```
dbListResults(conn, ...)
```

Arguments

conn	A DBIConnection object, as produced by dbConnect .
...	Other parameters passed on to methods.

Value

a list. If no results are active, an empty list. If only a single result is active, a list with one element.

See Also

Other connection methods: [dbDisconnect](#); [dbExistsTable](#); [dbGetException](#); [dbGetQuery](#), [dbGetQuery](#), [DBIConnection](#)
[dbListFields](#); [dbListTables](#); [dbReadTable](#), [dbWriteTable](#); [dbRemoveTable](#); [dbSendQuery](#)

 dbListTables

List remote tables.

Description

This should, where possible, include temporary tables.

Usage

```
dbListTables(conn, ...)
```

Arguments

conn	A DBIConnection object, as produced by dbConnect .
...	Other parameters passed on to methods.

Value

a character vector. If no tables present, a character vector of length 0.

See Also

Other connection methods: [dbDisconnect](#); [dbExistsTable](#); [dbGetException](#); [dbGetQuery](#), [dbGetQuery](#), [DBIConnection](#)
[dbListFields](#); [dbListResults](#); [dbReadTable](#), [dbWriteTable](#); [dbRemoveTable](#); [dbSendQuery](#)

 dbReadTable

Copy data frames to and from database tables.

Description

[dbReadTable](#): database table -> data frame; [dbWriteTable](#): data frame -> database table.

Usage

```
dbReadTable(conn, name, ...)
```

```
dbWriteTable(conn, name, value, ...)
```

Arguments

conn	A DBIConnection object, as produced by dbConnect .
name	A character string specifying a DBMS table name.
...	Other parameters passed on to methods.
value	a data.frame (or coercible to data.frame).

Value

a data.frame.

Note

The translation of identifiers between R and SQL is done through calls to [make.names](#) and [make.db.names](#), but we cannot guarantee that the conversion is reversible. For details see [make.db.names](#).

See Also

Other connection methods: [dbDisconnect](#); [dbExistsTable](#); [dbGetException](#); [dbGetQuery](#), [dbGetQuery](#), [DBIConnection](#); [dbListFields](#); [dbListResults](#); [dbListTables](#); [dbRemoveTable](#); [dbSendQuery](#)

Examples

```
if (require("RSQLite")) {
  con <- dbConnect(RSQLite::SQLite(), ":memory:")

  dbWriteTable(con, "mtcars", mtcars[1:10, ])
  dbReadTable(con, "mtcars")

  dbDisconnect(con)
}
```

dbRemoveTable	<i>Remove a table from the database.</i>
---------------	--

Description

Executes the sql DROP TABLE name.

Usage

```
dbRemoveTable(conn, name, ...)
```

Arguments

conn	A DBIConnection object, as produced by dbConnect .
name	A character string specifying a DBMS table name.
...	Other parameters passed on to methods.

Value

a logical vector of length 1 indicating success or failure.

See Also

Other connection methods: [dbDisconnect](#); [dbExistsTable](#); [dbGetException](#); [dbGetQuery](#), [dbGetQuery](#), [DBIConnection](#), [dbListFields](#); [dbListResults](#); [dbListTables](#); [dbReadTable](#), [dbWriteTable](#); [dbSendQuery](#)

 dbSendQuery

Execute a statement on a given database connection.

Description

The function `dbSendQuery` only submits and synchronously executes the SQL statement to the database engine. It does *not* extract any records — for that you need to use the function [dbFetch](#), and then you must call [dbClearResult](#) when you finish fetching the records you need.

Usage

```
dbSendQuery(conn, statement, ...)
```

Arguments

<code>conn</code>	A DBIConnection object, as produced by dbConnect .
<code>statement</code>	a character vector of length 1 containing SQL.
<code>...</code>	Other parameters passed on to methods.

Value

An object that inherits from [DBIResult](#). If the statement generates output (e.g., a SELECT statement) the result set can be used with [fetch](#) to extract records.

Side Effects

The statement is submitted for synchronous execution to the server connected through the `conn` object. The DBMS executes the statement, possibly generating vast amounts of data. Where these data reside is driver-specific: some drivers may choose to leave the output on the server and transfer them piecemeal to R, others may transfer all the data to the client – but not necessarily to the memory that R manages. See the individual drivers' [dbSendQuery](#) method for implementation details.

See Also

Other connection methods: [dbDisconnect](#); [dbExistsTable](#); [dbGetException](#); [dbGetQuery](#), [dbGetQuery](#), [DBIConnection](#), [dbListFields](#); [dbListResults](#); [dbListTables](#); [dbReadTable](#), [dbWriteTable](#); [dbRemoveTable](#)

Examples

```

if (require("RSQLite")) {
con <- dbConnect(RSQLite::SQLite(), ":memory:")

dbWriteTable(con, "mtcars", mtcars)
res <- dbSendQuery(con, "SELECT * FROM mtcars WHERE cyl = 4;")
dbFetch(res)
dbClearResult(res)

dbDisconnect(con)
}

```

make.db.names

Make R identifiers into legal SQL identifiers.

Description

These methods are DEPRECATED. Please use [dbQuoteIdentifier](#) (or possibly [dbQuoteString](#)) instead.

Usage

```

make.db.names(dbObj, snames, keywords = .SQL92Keywords, unique = TRUE,
  allow.keywords = TRUE, ...)

make.db.names.default(snames, keywords = .SQL92Keywords, unique = TRUE,
  allow.keywords = TRUE)

isSQLKeyword(dbObj, name, keywords = .SQL92Keywords, case = c("lower",
  "upper", "any")[3], ...)

isSQLKeyword.default(name, keywords = .SQL92Keywords, case = c("lower",
  "upper", "any")[3])

```

Arguments

dbObj	any DBI object (e.g., DBIDriver).
snames	a character vector of R identifiers (symbols) from which we need to make SQL identifiers.
keywords	a character vector with SQL keywords, by default it's .SQL92Keywords defined by the DBI.
unique	logical describing whether the resulting set of SQL names should be unique. Its default is TRUE. Following the SQL 92 standard, uniqueness of SQL identifiers is determined regardless of whether letters are upper or lower case.
allow.keywords	logical describing whether SQL keywords should be allowed in the resulting set of SQL names. Its default is TRUE

name	a character vector with database identifier candidates we need to determine whether they are legal SQL identifiers or not.
case	a character string specifying whether to make the comparison as lower case, upper case, or any of the two. it defaults to any.
...	any other argument are passed to the driver implementation.

Details

The algorithm in `make.db.names` first invokes `make.names` and then replaces each occurrence of a dot “.” by an underscore “_”. If `allow.keywords` is `FALSE` and identifiers collide with SQL keywords, a small integer is appended to the identifier in the form of “_n”.

The set of SQL keywords is stored in the character vector `.SQL92Keywords` and reflects the SQL ANSI/ISO standard as documented in "X/Open SQL and RDA", 1994, ISBN 1-872630-68-8. Users can easily override or update this vector.

Value

`make.db.names` returns a character vector of legal SQL identifiers corresponding to its `snames` argument.

`SQLKeywords` returns a character vector of all known keywords for the database-engine associated with `dbObj`.

`isSQLKeyword` returns a logical vector parallel to `name`.

Bugs

The current mapping is not guaranteed to be fully reversible: some SQL identifiers that get mapped into R identifiers with `make.names` and then back to SQL with `make.db.names` will not be equal to the original SQL identifiers (e.g., compound SQL identifiers of the form `username.tablename` will lose the dot “.”).

References

The set of SQL keywords is stored in the character vector `.SQL92Keywords` and reflects the SQL ANSI/ISO standard as documented in "X/Open SQL and RDA", 1994, ISBN 1-872630-68-8. Users can easily override or update this vector.

SQL

SQL quoting.

Description

This set of classes and generics make it possible to flexibly deal with SQL escaping needs. By default, any user supplied input to a query should be escaped using either `dbQuoteIdentifier` or `dbQuoteString` depending on whether it refers to a table or variable name, or is a literal string.

Usage

```
SQL(x)

dbQuoteIdentifier(conn, x, ...)

dbQuoteString(conn, x, ...)
```

Arguments

x	A character vector to label as being escaped SQL.
conn	A subclass of <code>DBIConnection</code> , representing an active connection to an DBMS.
...	Other arguments passed on to methods. Not otherwise used.

Details

The `SQL` class has associated `SQL()` constructor function. This class is used to prevent double escaping of SQL strings, and to make it possible to tell DBI functions that you've done the escaping yourself.

Implementation notes

DBI provides default methods for SQL-92 compatible quoting. If the database uses a different convention, you will need to provide your own methods. Note that because of the way that S4 dispatch finds methods and because `SQL` inherits from `character`, if you implement (e.g.) a method for `dbQuoteString(MyConnection, character)`, you will also need to implement `dbQuoteString(MyConnection, SQL)` - this should simply return `x` unchanged.

Examples

```
# Create a subclass of DBI connection since it's virtual
MockConnection <- setClass("MockConnection", "DBIConnection")
conn <- MockConnection()

# Quoting ensures that arbitrary input is safe for use in a query
name <- "Robert'); DROP TABLE Students;--"
dbQuoteString(conn, name)
dbQuoteIdentifier(conn, name)

# SQL vectors are always passed through as is
var_name <- SQL("select")
var_name

dbQuoteIdentifier(conn, var_name)
dbQuoteString(conn, var_name)

# This mechanism is used to prevent double escaping
dbQuoteString(conn, dbQuoteString(conn, name))
```

transactions

Begin/commit/rollback SQL transactions

Description

Not all database engines implement transaction management, in which case these methods should not be implemented for the specific [DBIConnection](#) subclass.

Usage

```
dbBegin(conn, ...)
```

```
dbCommit(conn, ...)
```

```
dbRollback(conn, ...)
```

Arguments

conn	A DBIConnection object, as produced by dbConnect .
...	Other parameters passed on to methods.

Value

a logical indicating whether the operation succeeded or not.

Side Effects

The current transaction on the connections con is committed or rolled back.

Examples

```
## Not run:
ora <- dbDriver("Oracle")
con <- dbConnect(ora)
rs <- dbSendQuery(con,
  "delete * from PURGE as p where p.wavelength<0.03")
if(dbGetInfo(rs, what = "rowsAffected") > 250){
  warning("dubious deletion -- rolling back transaction")
  dbRollback(con)
}

## End(Not run)
```

Index

- dbBegin (transactions), 25
- dbClearResult, 2, 3, 9, 11–14, 21
- dbColumnInfo, 3, 3, 9, 12–14, 18
- dbCommit (transactions), 25
- dbConnect, 4, 6, 8, 9, 11, 18–21, 25
- dbDataType, 5
- dbDataType, DBIObject-method (dbDataType), 5
- dbDisconnect, 4, 6, 8, 10, 11, 18–21
- dbDriver, 6
- dbDriver, character-method (dbDriver), 6
- dbExistsTable, 6, 7, 10, 11, 18–21
- dbFetch, 3, 8, 12–14, 21
- dbFetch, DBIResult-method (dbFetch), 8
- dbGetException, 6, 8, 9, 11, 18–21
- dbGetInfo, 10, 12, 13, 16, 17
- dbGetQuery, 6, 8, 10, 11, 18–21
- dbGetQuery, DBIConnection, character-method (dbGetQuery), 11
- dbGetRowCount, 3, 9, 12, 13, 14
- dbGetRowCount, DBIResult-method (dbGetRowCount), 12
- dbGetRowsAffected, 3, 9, 12, 12, 13, 14
- dbGetRowsAffected, DBIResult-method (dbGetRowsAffected), 12
- dbGetStatement, 3, 9, 12, 13, 13, 14
- dbGetStatement, DBIResult-method (dbGetStatement), 13
- dbHasCompleted, 3, 9, 11–13, 13
- dbHasCompleted, DBIResult-method (dbHasCompleted), 13
- dbiCheckCompliance, 14
- DBIConnection, 4, 6, 8–11, 17–21, 24, 25
- DBIConnection-class, 14
- DBIDriver, 4, 5, 10, 17
- DBIDriver-class, 15
- DBIObject, 10, 17
- DBIObject-class, 15
- DBIResult, 2, 3, 8, 10, 12, 13, 17, 18, 21
- DBIResult-class, 16
- dbIsValid, 10, 17
- dbListConnections, 17
- dbListFields, 6, 8, 10, 11, 18, 19–21
- dbListResults, 6, 8, 10, 11, 18, 18, 19–21
- dbListTables, 6, 8, 10, 11, 18, 19, 19, 20, 21
- dbQuoteIdentifier, 22
- dbQuoteIdentifier (SQL), 23
- dbQuoteIdentifier, DBIConnection, character-method (SQL), 23
- dbQuoteIdentifier, DBIConnection, SQL-method (SQL), 23
- dbQuoteString, 22
- dbQuoteString (SQL), 23
- dbQuoteString, DBIConnection, character-method (SQL), 23
- dbQuoteString, DBIConnection, SQL-method (SQL), 23
- dbReadTable, 6, 8, 10, 11, 18, 19, 19, 21
- dbRemoveTable, 6, 8, 10, 11, 18–20, 20, 21
- dbRollback (transactions), 25
- dbSendQuery, 6, 8, 10, 11, 18–21, 21
- dbUnloadDriver (dbDriver), 6
- dbWriteTable, 6, 8, 10, 11, 18, 19, 21
- dbWriteTable (dbReadTable), 19
- fetch, 3, 11–14, 21
- fetch (dbFetch), 8
- isSQLKeyword, 5
- isSQLKeyword (make.db.names), 22
- isSQLKeyword, DBIObject, character-method (make.db.names), 22
- isSQLKeyword.default (make.db.names), 22
- make.db.names, 5, 20, 22, 23
- make.db.names, DBIObject, character-method (make.db.names), 22
- make.db.names.default (make.db.names), 22

`make.names`, [20](#)

`on.exit`, [11](#)

`show`, SQL-method (SQL), [23](#)

SQL, [23](#)

SQL-class (SQL), [23](#)

SQLKeywords (make.db.names), [22](#)

SQLKeywords, DBIObject-method
(make.db.names), [22](#)

SQLKeywords, missing-method
(make.db.names), [22](#)

summary, [16](#)

transactions, [25](#)