# Package 'RMySQL'

August 29, 2013

**Version** 0.9-3

**Date** 2012-01-17

**Title** R interface to the MySQL database

**Author** David A. James and Saikat DebRoy

**Maintainer** Jeffrey Horner <jeffrey.horner@gmail.com>

**Description** Database interface and MySQL driver for R. This version
complies with the database interface definition as implemented in the package DBI 0.2-2.

**LazyLoad** true

**Depends** R (>= 2.8.0), methods, DBI (>= 0.2-2), utils

**License** GPL-2

**URL** http://biostat.mc.vanderbilt.edu/RMySQL, https://github.com/jeffreyhorner/RMySQL

**Collate** S4R.R zzz.R MySQLSupport.R dbObjectId.R MySQL.R

**Repository** CRAN

**Date/Publication** 2012-01-17 18:10:41

**NeedsCompilation** yes

## R topics documented:

---

`RMySQL-package` *R interface to the MySQL database*

---

## Description

The functions in this package allow you interact with one or more MySQL databases from R.

## Overview

A typical usage of the R-MySQL interface is:

1. Connect and authenticate to one or more MySQL databases:

```
con <- dbConnect(MySQL(), group = "lasers")
con2 <- dbConnect(MySQL(), user="opto", password="pure-light",
                  dbname="lasers", host="merced")
```

2. List tables and fields in a table:

```
dbListTables(con)
dbListFields(con, "table_name")
```

3. Import and export data.frames:

```
d <- dbReadTable(con, "WL")
dbWriteTable(con, "WL2", a.data.frame)         ## table from a data.frame
dbWriteTable(con, "test2", "~/data/test2.csv") ## table from a file
```

4. Run an arbitrary SQL statement and extract all its output (returns a data.frame):

```
dbGetQuery(con, "select count(*) from a_table")
dbGetQuery(con, "select * from a_table")
```

5. Run an SQL statement and extract its output in pieces (returns a result set):

```
rs <- dbSendQuery(con, "select * from WL where width_nm between 0.5 and 1")
d1 <- fetch(rs, n = 10000)
d2 <- fetch(rs, n = -1
```

6. Run multiple SQL statements and process the various result sets (note the `client.flag` value in the `dbConnect` call):

```
con <- dbConnection(MySQL(), dbname = "rs-dbi",
          client.flag = CLIENT_MULTI_STATEMENTS)
script  <- paste("select * from WL where width_nm between 0.5 and 1"
                 "select * from lasers_id  where id LIKE 'AL100
                 sep = ";")
rs1 <- dbSendQuery(con, script)
d1 <- fetch(rs1, n = -1)
if(dbMoreResults(con)){
   rs2 <- dbNextResult(con)
   d2 <- fetch(rs2, n=-1)
}
```

7. Get meta-information on a connection (thread-id, etc.):

```
summary(MySQL(), verbose = TRUE)
summary(con, verbose = TRUE)
summary(rs, verbose = TRUE)
dbListConnections(MySQL())
dbListResultSets(con)
dbHasCompleted(rs)
```

8. Close connections:

```
dbDisconnect(con)
dbDisconnect(con2)
```

## Data mappings between MySQL and R

MySQL tables are read into R as data.frames, but without coercing character or logical data into factors. Similarly while exporting data.frames, factors are exported as character vectors.

Integer columns are usually imported as R integer vectors, except for cases such as BIGINT or UNSIGNED INTEGER which are coerced to R's double precision vectors to avoid truncation (currently R's integers are signed 32-bit quantities).

Time variables are imported/exported as character data, so you need to convert these to your favorite date/time representation.

Currently there are no facilities to import/export BLOBs.

### RDBMS tables, data.frames, and data types

Tables in a relational database are only superficially similar to R's data.frames (e.g., tables as unordered sets of rows compared to data.frames as ordered sets, tables having referential constraints, indexes, and so on.)

### User authentication

Although you can specify user authentication parameters (user, password, database, and host) in the call to dbConnect, the preferred method to pass these parameters to the server is through a MySQL default.file, e.g., '\$HOME/.my.cnf' (or 'c:/my.cnf' under Windows). The MySQL dbConnect method parses the default.file=$HOME/.my.cnf to initialize connections to MySQL databases. This file consists of zero or more named sections each starting with a line of the form [section-name]; each section includes zero or more MySQL variable declaration per line, such as, user=, password=, host=, etc. For instance,

```
$ cat $HOME/.my.cnf
# this is a comment
; this is also a comment
[client]
user = dj
host = localhost

[rs-dbi]
database = s-data

[lasers]
user = opto
database = opto
password = pure-light
host = merced
...
[iptraffic]
host = data
database = iptraffic
```

This file should be readable only by you. RMySQL always initializes connection values from the [client] and [rs-dbi] sections, but you may define you own project-specific sections (as in the example above) to tailor its environment; if the same parameter appears in multiple sections (e.g., in client and rs-dbi), the last (closer to the bottom) occurrence is used.

If you define a section, for instance, [iptraffic], then instead of including all these parameters in the call to dbConnect, you simply supply the name of the group, e.g., dbConnect(MySQL(), group = "iptraffic").

In addition to user, password, host, and dbname, you may specify any other connection parameters, e.g., port, socket. See the MySQL documentation for details.

Lastly, you may specify an alternate default.file, e.g., dbConnect(MySQL(), group="iptraffic", default.file="rc

## References

See stat.bell-labs.com/RS-DBI for more details on the R/S-Plus database interface.

See the documentation at the MySQL Web site http://www.mysql.com for details.

## Author(s)

David A. James <dj@bell-labs.com> Saikat DebRoy <saikat@stat.wisc.edu>

## See Also

On database managers:

dbDriver dbUnloadDriver

On connections, SQL statements and resultSets:

dbConnect dbDisconnect dbSendQuery dbGetQuery fetch dbClearResult

On transaction management:

dbCommit dbRollback

On meta-data:

summary dbGetInfo dbGetDBIVersion dbListTables dbListConnections dbListResults dbColumnInfo dbGetException dbGetStatement dbHasCompleted dbGetRowCount

## Examples

```
## Not run:
# create a MySQL instance and create one connection.
> m <- dbDriver("MySQL")  ## or MySQL()
<MySQLDriver:(4378)>

# open the connection using user, passsword, etc., as
# specified in the "[iptraffic]" section of the
# configuration file \file{$HOME/.my.cnf}
> con <- dbConnect(m, group = "iptraffic")
> rs <- dbSendQuery(con, "select * from HTTP_ACCESS where IP_ADDRESS = '127.0.0.1'")
> df <- fetch(rs, n = 50)
> dbHasCompleted(rs)
[1] FALSE
> df2 <- fetch(rs, n = -1)
> dbHasCompleted(rs)
[1] TRUE
> dbClearResult(rs)
> dim(dbGetQuery(con, "show tables"))
[1] 74   1
```

```
> dbListTables(con)

## End(Not run)
```

---

dbApply *Apply R functions to remote groups of DBMS rows (experimental)*

---

## Description

Applies R functions to groups of remote DBMS rows without bringing an entire result set all at once. The result set is expected to be sorted by the grouping field.

## Usage

```
dbApply(res, ...)
```

## Arguments

| | |
|---|---|
| res | a result set (see dbSendQuery). |
| ... | any additional arguments to be passed to FUN. |

## Details

dbApply This generic is meant to handle somewhat gracefully(?) large amounts of data from the DBMS by bringing into R manageable chunks; the idea is that the data from individual groups can be handled by R, but not all the groups at the same time.

Currently, only the MySQL driver implements a method (see the helper function mysqlDBApply) for this generic function.

## Value

A list with as many elements as there were groups in the result set.

## See Also

MySQL mysqlDBApply dbSendQuery fetch

## Examples

```
## Not run:
## compute quanitiles for each network agent
con <- dbConnect(MySQL(), group="vitalAnalysis")
rs <- dbSendQuery(con,
             "select Agent, ip_addr, DATA from pseudo_data order by Agent")
out <- dbApply(rs, INDEX = "Agent",
        FUN = function(x, grp) quantile(x$DATA, names=FALSE))

## End(Not run)
```

dbApply-methods *Apply R/S-Plus functions to remote groups of DBMS rows (experimental)*

## Description

Applies R/S-Plus functions to groups of remote DBMS rows without bringing an entire result set all at once. The result set is expected to be sorted by the grouping field.

## Methods

**res** a MySQL result set (see dbSendQuery).

**...** any additional arguments to be passed to FUN.

## References

See the Database Interface definition document DBI.pdf in the base directory of this package or http://stat.bell-labs.com/RS-DBI.

## See Also

MySQL mysqlDBApply dbSendQuery fetch

## Examples

```
## Not run:
## compute quanitiles for each network agent
con <- dbConnect(MySQL(), group="vitalAnalysis")
rs <- dbSendQuery(con,
             "select Agent, ip_addr, DATA from pseudo_data order by Agent")
out <- dbApply(rs, INDEX = "Agent",
        FUN = function(x, grp) quantile(x$DATA, names=FALSE))

## End(Not run)
```

dbBuildTableDefinition *Build the SQL CREATE TABLE definition as a string*

## Description

Build the SQL CREATE TABLE definition as a string for the input data.frame

## Usage

```
dbBuildTableDefinition(dbObj, name, obj, field.types = NULL, row.names = TRUE, ...)
```

### Arguments

| | |
|---|---|
| dbObj | any DBI object (used only to dispatch according to the engine (e.g., MySQL, Oracle, PostgreSQL, SQLite) |
| name | name of the new SQL table |
| obj | an R object coerceable to data.frame for which we want to create a table |
| field.types | optional named list of the types for each field in obj |
| row.names | logical, should row.name of value be exported as a row_names field? Default is TRUE |
| ... | reserved for future use |

### Details

The output SQL statement is a simple CREATE TABLE with suitable for dbGetQuery

### Value

An SQL string

### References

See the Database Interface definition document DBI.pdf in the base directory of this package or http://stat.bell-labs.com/RS-DBI.

### See Also

MySQL, dbConnect, dbSendQuery, dbGetQuery, fetch, dbCommit, dbGetInfo, dbReadTable.

---

## dbCallProc-methods *Call an SQL stored procedure*

---

### Description

Not yet implemented.

### Methods

**conn** a MySQLConnection object.

**...** additional arguments are passed to the implementing method.

### References

See the Database Interface definition document DBI.pdf in the base directory of this package or http://stat.bell-labs.com/RS-DBI.

### See Also

MySQL, dbConnect, dbSendQuery, dbGetQuery, fetch, dbCommit, dbGetInfo, dbReadTable.

dbCommit-methods *DBMS Transaction Management*

## Description

Commits or roll backs the current transaction in an MySQL connection

## Methods

**conn** a MySQLConnection object, as produced by the function dbConnect.

**...** currently unused.

## References

See the Database Interface definition document DBI.pdf in the base directory of this package or http://stat.bell-labs.com/RS-DBI.

## See Also

MySQL, dbConnect, dbSendQuery, dbGetQuery, fetch, dbCommit, dbGetInfo, dbReadTable.

## Examples

```
## Not run:
drv <- dbDriver("MySQL")
con <- dbConnect(drv, group = "group")
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)
```

dbConnect-methods *Create a connection object to an MySQL DBMS*

## Description

These methods are straight-forward implementations of the corresponding generic functions.

## Methods

**drv** an object of class MySQLDriver, or the character string "MySQL" or an MySQLConnection.

**conn** an MySQLConnection object as produced by dbConnect.

**username** string of the MySQL login name or NULL. If NULL or the empty string "", the current user is assumed.

**password** string with the MySQL password or NULL. If NULL, only entries in the user table for the users that have a blank (empty) password field are checked for a match.

**dbname** string with the database name or NULL. If NOT NULL, the connection sets the default database to this value.

**host** string identifying the host machine running the MySQL server or NULL. If NULL or the string "localhost", a connection to the local host is assumed.

**unix.socket** (optional) string of the unix socket or named pipe.

**port** (optional) integer of the TCP/IP default port.

**client.flag** (optional) integer setting various MySQL client flags. See the MySQL manual for details.

**group** string identifying a section in the default.file to use for setting authentication parameters (see MySQL.)

**default.file** string of the filename with MySQL client options. Defaults to $HOME/.my.cnf

**...** Currently unused.

## Side Effects

A connection between R/S-Plus and an MySQL server is established. The current implementation supports up to 100 simultaneous connections.

## References

See the Database Interface definition document DBI.pdf in the base directory of this package or http://stat.bell-labs.com/RS-DBI.

## See Also

MySQL, dbConnect, dbSendQuery, dbGetQuery, fetch, dbCommit, dbGetInfo, dbReadTable.

## Examples

```
## Not run:
# create an MySQL instance and create one connection.
drv <- dbDriver("MySQL")

# open the connection using user, passsword, etc., as
con <- dbConnect(drv, group = "rs-dbi")

# Run an SQL statement by creating first a resultSet object
rs <- dbSendQuery(con, statement = paste(
                      "SELECT w.laser_id, w.wavelength, p.cut_off",
```

```
                    "FROM WL w, PURGE P",
                    "WHERE w.laser_id = p.laser_id",
                    "SORT BY w.laser_id")
# we now fetch records from the resultSet into a data.frame
data <- fetch(rs, n = -1)   # extract all rows
dim(data)

## End(Not run)
```

---

dbDataType-methods *Determine the SQL Data Type of an S object*

---

## Description

This method is a straight-forward implementation of the corresponding generic function.

## Arguments

| | |
|---|---|
| dbObj | any MySQLObject object, e.g., MySQLDriver, MySQLConnection, MySQLResult. |
| obj | R/S-Plus object whose SQL type we want to determine. |
| ... | any other parameters that individual methods may need. |

## References

See the Database Interface definition document DBI.pdf in the base directory of this package or http://stat.bell-labs.com/RS-DBI.

## See Also

isSQLKeyword make.db.names

## Examples

```
## Not run:
data(quakes)
drv <- dbDriver("MySQL")
sql.type <- dbDataType(drv, quakes)

## End(Not run)
```

dbDriver-methods *MySQL implementation of the Database Interface (DBI) classes and drivers*

## Description

MySQL driver initialization and closing

## Methods

**drvName** character name of the driver to instantiate.

**drv** an object that inherits from MySQLDriver as created by dbDriver.

**max.con** optional integer requesting the maximum number of simultanous connections (may be up to 100).

**fetch.default.rec** default number of records to retrieve per fetch. Default is 500. This may be overridden in calls to fetch with the n= argument.

**force.reload** optional logical used to force re-loading or recomputing the size of the connection table. Default is FALSE.

**...** currently unused.

## References

See the Database Interface definition document DBI.pdf in the base directory of this package or http://stat.bell-labs.com/RS-DBI.

## See Also

MySQL, dbConnect, dbSendQuery, dbGetQuery, fetch, dbCommit, dbGetInfo, dbListTables, dbReadTable.

## Examples

```
## Not run:
# create an MySQL instance and set 10000 of rows per fetch.
m <- dbDriver("MySQL", fetch.default.records=10000)

con <- dbConnect(m, username="usr", password = "pwd",
           dbname = "iptraffic")
rs <- dbSubmitQuery(con,
         "select * from HTTP_ACCESS where IP_ADDRESS = '127.0.0.1'")
df <- fetch(rs, n = 50)
df2 <- fetch(rs, n = -1)
dbClearResult(rs)

pcon <- dbConnect(p, group = "wireless")
dbListTables(pcon)

## End(Not run)
```

dbEscapeStrings *Escape SQL-special characters in strings*

## Description

is expected to be sorted by the grouping field.

## Usage

```
dbEscapeStrings(con, strings, ...)
```

## Arguments

| | |
|---|---|
| con | a connection object (see dbConnect). |
| strings | a character vector. |
| ... | any additional arguments to be passed to the dispatched method. |

## Details

dbEscapeStrings

Currently, only the MySQL driver implements this method.

## Value

A character vector with SQL special characters properly escaped.

## See Also

MySQL dbSendQuery fetch

## Examples

```
## Not run:
tmp <- sprintf("select * from emp where lname = %s", "O'Reilly")
sql <- dbEscapeString(con, tmp)
dbGetQuery(con, sql)

## End(Not run)
```

## dbEscapeStrings-methods *Escape a Character Vector According to SQL rules*

### Description

Escape SQL-special characters in a character vector according to MySQL rules.

### Methods

**con = "MySQLConnection", strings = "character"** This method encodes the `strings` character vector according to the MySQL escape rules and taking into consideration the character set used by the connection (each MySQL connection may be set to use different character sets). Note that the `RMySQL` package currently does not deal with character set conversions – it uses whatever character encoding the R session is using, but the MySQL runtime library handles this transparently.

**...** currently unused.

### See Also

`MySQL`, `dbConnect`, `dbSendQuery`, `dbGetQuery`, `fetch`, `dbNextResult`, `dbCommit`, `dbGetInfo`, `dbReadTable`.

### Examples

```
## Not run:
con <- dbConnect(MySQL(),
          dbname = "rs-dbi",
          client.flag=CLIENT_MULTI_STATEMENTS)
tmp <- sprintf("select * from emp where lname =
sql <- dbEscapeString(con, tmp)
dbGetQuery(con, sql)

## End(Not run)
```

## dbGetInfo-methods *Database interface meta-data*

### Description

These methods are straight-forward implementations of the corresponding generic functions.

### Methods

**dbObj** any object that implements some functionality in the R/S-Plus interface to databases (a driver, a connection or a result set).

**res** an `MySQLResult`.

**...** currently not being used.

### References

See the Database Interface definition document `DBI.pdf` in the base directory of this package or http://stat.bell-labs.com/RS-DBI.

### See Also

`MySQL`, `dbDriver`, `dbConnect`, `dbSendQuery`, `dbGetQuery`, `fetch`, `dbCommit`, `dbGetInfo`, `dbListTables`, `dbReadTable`.

### Examples

```
## Not run:
drv <- dbDriver("MySQL")
con <- dbConnect(drv, group = "wireless")

dbListTables(con)

rs <- dbSendQuery(con, query.sql)
dbGetStatement(rs)
dbHasCompleted(rs)

info <- dbGetInfo(rs)
names(dbGetInfo(drv))

# DBIConnection info
names(dbGetInfo(con))

# DBIResult info
names(dbGetInfo(rs))

## End(Not run)
```

---

## dbListTables-methods *List items from an MySQL DBMS and from objects*

---

### Description

These methods are straight-forward implementations of the corresponding generic functions.

## Methods

**drv** an MySQLDriver.

**conn** an MySQLConnection.

**name** a character string with the table name.

**...** currently not used.

## References

See the Database Interface definition document DBI.pdf in the base directory of this package or http://stat.bell-labs.com/RS-DBI.

## See Also

MySQL, dbGetInfo, dbColumnInfo, dbDriver, dbConnect, dbSendQuery

## Examples

```
## Not run:
drv <- dbDriver("MySQL")
# after working awhile...
for(con in dbListConnections(drv)){
   dbGetStatement(dbListResults(con))
}

## End(Not run)
```

---

# dbNextResult *Fetch next result set from an SQL script or stored procedure (experimental)*

---

## Description

Fetches the next result set from the output of a multi-statement SQL script or stored procedure; checkes whether there are additonal result sets to process.

## Usage

```
dbNextResult(con, ...)
dbMoreResults(con, ...)
```

## Arguments

| | |
|---|---|
| con | a connection object (see dbConnect). |
| ... | any additional arguments to be passed to the dispatched method |

## Details

SQL scripts (i.e., multiple SQL statements separated by ';') and stored procedures oftentimes generate multiple result sets. These DBI generic functions provide a means to process them sequentially.

`dbNextResult` fetches the next result from the sequence of pending results sets; `dbMoreResults` returns a logical to indicate whether there are additional results to process.

## Value

`dbNextResult` returns a result set or `NULL`.

`dbMoreResults` returns a logical specifying whether or not there are additional result sets to process in the connection.

## Note

Currently only the `MySQL` driver implements these methods. See 'methods?dbNextMethod'.

## See Also

`MySQL` `dbConnect` `dbSendQuery` `fetch`

## Examples

```
## Not run:
rs1 <- dbSendQuery(con,
         paste(
             "select Agent, ip_addr, DATA from pseudo_data order by Agent",
             "select * from Agent_name",
              sep = ";")
         )
x1 <- fetch(rs1, n = -1)
if(dbMoreResults(con)){
   rs2 <- dbNextResult(con)
   x2 <- fetch(rs2, n = -1)
}

## End(Not run)
```

---

`dbNextResult-methods` *Fetch Next Result Set from Multiple Statements or Stored Procedures*

---

## Description

`dbMoreResults` checks whether there are additional result sets for processing. `dbNextResult` fetches the next result set.

## Methods

These MySQL methods provide functionality to sequentially extract multiple results produced by SQL scripts or stored procedures.

In the case of stored procedures invoked with `CALL`, the first result set indicates the call status, and output data (if any) are return as additional result sets.

a MySQL connection object.

## Note

**con = "MySQLConnection"** MySQL supports SQL scripts (a single string with multiple statements terminated by ';') from version 4.1.1 onwards and stored procedures from version 5.0.

To process SQL scripts on a MySQL connection, the connection must be created using the `CLIENT_MULTI_STATEMENTS`. In addition, to process stored procedures that return one or more result sets, the connection must be created using the `CLIENT_MULTI_RESULTS` client flag.

For simplicity, use `CLIENT_MULTI_STATEMENTS` for working with either SQL scripts or stored procedures. For more details, read on.

More precisely, to execute multiple statements the connection needs `CLIENT_MULTI_STATEMENTS`; this in turn automatically enables `CLIENT_MULTI_RESULTS` for *fetching* of multiple output results. On the other hand, the client flag `CLIENT_MULTI_RESULTS` by itself enables stored procedures to return one or more results. See the MySQL documentation in www.mysql.com for full details.

## See Also

MySQL, dbConnect, dbSendQuery, dbHasCompleted, fetch, dbCommit, dbGetInfo, dbReadTable.

## Examples

```
## Not run:
con <- dbConnect(MySQL(),
          dbname = "rs-dbi",
          client.flag=CLIENT_MULTI_STATEMENTS)
sql.script <- paste(
   "select * from abc",
   "select * def",
   collapse = ";")

rs1 <- dbSendQuery(con, sql.script)
data1 <- fetch(rs1, n = -1)

if(dbMoreResults(con)){
   rs2 <- dbNextResult(con)
   ## you could use dbHasCompleted(rs2) to determine whether
   ## rs2 is a select-like that generates output or not.
   data2 <- fetch(rs2, n = -1)
   }

## End(Not run)
```

dbObjectId-class *Class dbObjectId*

## Description

A helper (mixin) class to provide external references in an R/S-Plus portable way.

## Objects from the Class

A virtual Class: No objects may be created from it.

## Slots

`Id`: Object of class `"integer"` this is an integer vector holding an opaque reference into a C struct (may or may not be a C pointer, may or may not have length one).

## Methods

**coerce** `signature(from = "dbObjectId", to = "integer")`: ...

**coerce** `signature(from = "dbObjectId", to = "numeric")`: ...

**coerce** `signature(from = "dbObjectId", to = "character")`: ...

**format** `signature(x = "dbObjectId")`: ...

**print** `signature(x = "dbObjectId")`: ...

**show** `signature(object = "dbObjectId")`: ...

## Note

A cleaner mechanism would use external references, but historically this class has existed mainly for R/S-Plus portability.

## Examples

```
## Not run:
  pg <- dbDriver("PostgreSQL")
  con <- dbConnect(pg, "user", "password")
  is(pg, "dbObjectId")   ## True
  is(con, "dbObjectId")  ## True
  isIdCurrent(con)       ## True
  q("yes")
  $ R
  isIdCurrent(con)       ## False

## End(Not run)
```

dbReadTable-methods *Convenience functions for Importing/Exporting DBMS tables*

## Description

These functions mimic their R/S-Plus counterpart `get`, `assign`, `exists`, `remove`, and `objects`, except that they generate code that gets remotely executed in a database engine.

## Value

A `data.frame` in the case of `dbReadTable`; otherwise a logical indicating whether the operation was successful.

## Methods

**conn** an `MySQLConnection` database connection object.

**name** a character string specifying a table name.

**value** a data.frame (or coercible to data.frame).

**row.names** in the case of `dbReadTable`, this argument can be a string or an index specifying the column in the DBMS table to be used as `row.names` in the output data.frame (a `NULL`, `""`, or 0 specifies that no column should be used as `row.names` in the output).

In the case of `dbWriteTable`, this argument should be a logical specifying whether the `row.names` should be output to the output DBMS table; if `TRUE`, an extra field whose name will be whatever the R/S-Plus identifier `"row.names"` maps to the DBMS (see `make.db.names`).

**overwrite** a logical specifying whether to overwrite an existing table or not. Its default is `FALSE`.

**append** a logical specifying whether to append to an existing table in the DBMS. Its default is `FALSE`.

**allow.keywords** `dbWriteTable` accepts a logical `allow.keywords` to allow or prevent MySQL reserved identifiers to be used as column names. By default it is `FALSE`.

**dots** optional arguments.

When `dbWriteTable` is used to import data from a file, you may optionally specify `header=`, `row.names=`, `col.names=`, `sep=`, `eol=`, `field.types=`, `skip=`, and `quote=`.

`header` is a logical indicating whether the first data line (but see `skip`) has a header or not. If missing, it value is determined following `read.table` convention, namely, it is set to TRUE if and only if the first row has one fewer field that the number of columns.

`row.names` is a logical to specify whether the first column is a set of row names. If missing its default follows the `read.table` convention.

`col.names` a character vector with column names (these names will be filtered with `make.db.names` to ensure valid SQL identifiers. (See also `field.types` below.)

`sep=` specifies the field separator, and its default is `','`.

`eol=` specifies the end-of-line delimiter, and its default is `'\n'`.

`skip` specifies number of lines to skip before reading the data, and it defaults to 0.

`field.types` is a list of named field SQL types where `names(field.types)` provide the new table's column names (if missing, field types are inferred using `dbDataType`).

## Note

Note that data.frames are only approximately analogous to tables (relations) in relational DBMS, and thus you should not expect complete agreement in their semantics. Tables in RDBMS are best thought of as *relations* with a number of constraints imposed by the relational database model, and data.frames, with their roots in statistical modeling, as self-contained "sequence of observations on some chosen variables" (Chambers and Hastie (1992), p.46). In particular the `data.frame` returned by `dbReadTable` only has primitive data, e.g., it does not coerce character data to factors. Also, column names in a data.frame are *not* guaranteed to be equal to the column names in a MySQL exported/imported table (e.g., by default MySQL reserved identifiers may not be used as column names — and with 218 keywords like `"BEFORE"`, `"DESC"`, and `"FROM"` the likelihood of name conflicts is not small.) Use `isSQLKeyword(con, names(value))` to check whether the data.frame names in `value` coincide with MySQL reserver words.

MySQL table names are *not* case sensitive, e.g., table names `ABC` and `abc` are considered equal.

## References

See the Database Interface definition document `DBI.pdf` in the base directory of this package or http://stat.bell-labs.com/RS-DBI.

## See Also

MySQL, mysqlImportFile, isSQLKeyword, dbDriver, dbConnect, dbSendQuery, dbGetQuery, fetch, dbCommit, dbGetInfo, dbListTables, dbReadTable.

## Examples

```
## Not run:
conn <- dbConnect("MySQL", group = "wireless")
if(dbExistsTable(con, "fuel_frame")){
   dbRemoveTable(conn, "fuel_frame")
   dbWriteTable(conn, "fuel_frame", fuel.frame)
}
if(dbExistsTable(conn, "RESULTS")){
   dbWriteTable(conn, "RESULTS", results2000, append = T)
else
   dbWriteTable(conn, "RESULTS", results2000)
}

## End(Not run)
```

---

`dbSendQuery-methods` *Execute a statement on a given database connection*

---

## Description

These methods are straight-forward implementations of the corresponding generic functions.

### Methods

**conn** an MySQLConnection object.

**statement** a character vector of length 1 with the SQL statement.

**res** an MySQLResult object.

**...** additional parameters.

### References

See the Database Interface definition document DBI.pdf in the base directory of this package or http://stat.bell-labs.com/RS-DBI.

### See Also

MySQL, dbDriver, dbConnect, fetch, dbCommit, dbGetInfo, dbReadTable.

### Examples

```
## Not run:
drv <- dbDriver("MySQL")
con <- dbConnect(drv, "usr", "password", "dbname")
res <- dbSendQuery(con, "SELECT * from liv25")
data <- fetch(res, n = -1)

## End(Not run)
```

---

## dbSetDataMappings-methods

*Set data mappings between MySQL and R/S-Plus*

---

### Description

Not yet implemented

### Methods

**res** a MySQLResult object as returned by dbSendQuery.

**flds** a data.frame with field descriptions as returned by dbColumnInfo.

**...** any additional arguments are passed to the implementing method.

### References

See the Database Interface definition document DBI.pdf in the base directory of this package or http://stat.bell-labs.com/RS-DBI.

### See Also

MySQL, dbSendQuery, fetch, dbColumnInfo.

## Examples

```
## Not run:
makeImage <- function(x) {
  .C("make_Image", as.integer(x), length(x))
}

res <- dbSendQuery(con, statement)
flds <- dbColumnInfo(res)
flds[3, "Sclass"] <- makeImage

dbSetDataMappings(rs, flds)

im <- fetch(rs, n = -1)

## End(Not run)
```

---

`fetch-methods` *Fetch records from a previously executed query*

---

## Description

This method is a straight-forward implementation of the corresponding generic function.

## Details

The RMySQL implementations retrieves only n records, and if n is missing it only returns up to fetch.default.rec as specified in the call to MySQL (500 by default).

## Methods

**res** an MySQLResult object.

**n** maximum number of records to retrieve per fetch. Use n = -1 to retrieve all pending records; use a value of n = 0 for fetching the default number of rows fetch.default.rec defined in the MySQL initialization invocation.

**...** currently not used.

## References

See the Database Interface definition document DBI.pdf in the base directory of this package or http://stat.bell-labs.com/RS-DBI.

## See Also

MySQL, dbConnect, dbSendQuery, dbGetQuery, dbClearResult, dbCommit, dbGetInfo, dbReadTable.

## Examples

```
## Not run:
drv <- dbDriver("MySQL")
con <- dbConnect(drv, user = "opto", password="pure-light",
                 host = "localhost", dbname="lasers")
res <- dbSendQuery(con, statement = paste(
                      "SELECT w.laser_id, w.wavelength, p.cut_off",
                      "FROM WL w, PURGE P",
                      "WHERE w.laser_id = p.laser_id",
                      "ORDER BY w.laser_id"))
# we now fetch the first 100 records from the resultSet into a data.frame
data1 <- fetch(res, n = 100)
dim(data1)

dbHasCompleted(res)

# let's get all remaining records
data2 <- fetch(res, n = -1)

## End(Not run)
```

---

`isIdCurrent` *Check whether a database handle object is valid or not*

---

## Description

Support function that verifies that an object holding a reference to a foreign object is still valid for communicating with the RDBMS

## Usage

```
isIdCurrent(obj)
```

## Arguments

| | |
|---|---|
| `obj` | any `dbObject` (e.g., `dbDriver`, `dbConnection`, `dbResult`). |

## Details

`dbObjects` are R/S-Plus remote references to foreign objects. This introduces differences to the object's semantics such as persistence (e.g., connections may be closed unexpectedly), thus this function provides a minimal verification to ensure that the foreign object being referenced can be contacted.

## Value

a logical scalar.

## See Also

dbDriver dbConnect dbSendQuery fetch

## Examples

```
## Not run:
cursor <- dbSendQuery(con, sql.statement)
isIdCurrent(cursor)

## End(Not run)
```

---

# make.db.names-methods *Make R/S-Plus identifiers into legal SQL identifiers*

---

## Description

These methods are straight-forward implementations of the corresponding generic functions.

## Methods

**dbObj** any MySQL object (e.g., MySQLDriver).

**snames** a character vector of R/S-Plus identifiers (symbols) from which we need to make SQL identifiers.

**name** a character vector of SQL identifiers we want to check against keywords from the DBMS.

**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

**keywords** a character vector with SQL keywords, by default it is .MySQLKeywords define in RMySQL. This may be overriden by users.

**case** a character string specifying whether to make the comparison as lower case, upper case, or any of the two. it defaults to any.

**...** currently not used.

## 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.

MySQL does add some keywords to the SQL 92 standard, they are listed in the .MySQLKeywords object.

See the Database Interface definition document DBI.pdf in the base directory of this package or http://stat.bell-labs.com/RS-DBI.

## See Also

MySQL, dbReadTable, dbWriteTable, dbExistsTable, dbRemoveTable, dbListTables.

## Examples

```
## Not run:
# This example shows how we could export a bunch of data.frames
# into tables on a remote database.

con <- dbConnect("MySQL", "user", "password")

export <- c("trantime.email", "trantime.print", "round.trip.time.email")
tabs <- make.db.names(export, unique = T, allow.keywords = T)

for(i in seq(along = export) )
   dbWriteTable(con, name = tabs[i],  get(export[i]))

## End(Not run)
```

---

# MySQL *Instantiate a MySQL client from the current R session*

---

## Description

This function creates and initializes a MySQL client. It returns an driver object that allows you to connect to one or several MySQL servers.

## Usage

```
MySQL(max.con = 16, fetch.default.rec = 500, force.reload = FALSE)
```

## Arguments

| | |
|---|---|
| max.con | maximum number of connections that are intended to have open at one time. There's no intrinic limit, since strictly speaking this limit applies to MySQL *servers*, but clients can have (at least in theory) more than this. Typically there are at most a handful of open connections, thus the internal RMySQL code uses a very simple linear search algorithm to manage its connection table. |
| fetch.default.rec | number of records to fetch at one time from the database. (The fetch method uses this number as a default.) |
| force.reload | should the client code be reloaded (reinitialize)? Setting this to TRUE allows you to change default settings. Notice that all connections should be closed before re-loading. |

## Details

This object is a singleton, that is, on subsequent invocations it returns the same initialized object.

This implementation allows you to connect to multiple host servers and run multiple connections on each server simultaneously.

## Value

An object `MySQLDriver` that extends `dbDriver` and `dbObjectId`. This object is required to create connections to one or several MySQL database engines.

## Side Effects

The R client part of the database communication is initialized, but note that connecting to the database engine needs to be done through calls to `dbConnect`.

## User authentication

The preferred method to pass authentication parameters to the server (e.g., user, password, host) is through the MySQL personal configuration file '`\$HOME/.my.cnf`' (or '`c:/my.cnf`' under Windows). Since specifying passwords on calls to `dbConnect` is a very bad idea (and so is specifying passwords through shell variables), the client code parses the configuration file '`\$HOME/.my.cnf`'; this file consists of zero or more sections, each starting with a line of the form `[section-name]`, for instance

\$ cat \$HOME/.my.cnf
\# this is a comment
[client]
user = dj
host = localhost

[rs-dbi]
database = s-data

[lasers]
user = opto
database = opto
password = pure-light
host = merced
...
[iptraffic]
host = data
database = iptraffic

This file should be readeable only by you. Inside each section, MySQL parameters may be specified one per line (e.g., `user = opto`). MySQL always considers default options from the `[client]` group for connecting to a server. To override or add additional options, R MySQL combines default options from the `[rs-dbi]` group, but you may specifiy you own group in the `dbConnect` call to tailor your environment. Note that to override options, you must place your group after the `[client]` group in configuration file.

For instance, if you define a group, say, [iptraffic], then instead of including all these parameters in the call to dbConnect, you simply supply the name of the group, e.g., dbConnect(mgr, group = "iptraffic").

The most important parameters are user, password, host, and dbname.

## References

See stat.bell-labs.com/RS-DBI for more details on the R/S-Plus database interface.

See the documentation at the MySQL Web site http://www.mysql.com for details.

## Note

Use the option database in place ofdbname in configuration files.

## Author(s)

David A. James

## See Also

On database managers:

dbDriver dbUnloadDriver

On connections, SQL statements and resultSets:

dbConnect dbDisconnect dbSendQuery dbGetQuery fetch dbClearResult

On transaction management:

dbCommit dbRollback

On meta-data:

summary dbGetInfo dbGetDBIVersion dbListTables dbListConnections dbListResults dbColumnInfo dbGetException dbGetStatement dbHasCompleted dbGetRowCount

## Examples

```
## Not run:
# create a MySQL instance and create one connection.
> m <- dbDriver("MySQL")
<MySQLDriver:(4378)>

# open the connection using user, passsword, etc., as
# specified in the "[iptraffic]" section of the
# configuration file \file{$HOME/.my.cnf}
> con <- dbConnect(m, group = "iptraffic")
> rs <- dbSendQuery(con, "select * from HTTP_ACCESS where IP_ADDRESS = '127.0.0.1'")
> df <- fetch(rs, n = 50)
> dbHasCompleted(rs)
[1] FALSE
> df2 <- fetch(rs, n = -1)
> dbHasCompleted(rs)
[1] TRUE
> dbClearResult(rs)
```

```
> dim(dbGetQuery(con, "show tables"))
[1] 74  1
> dbListTables(con)

## End(Not run)
```

---

`mysqlClientLibraryVersions` *MySQL Check for Compiled Versus Loaded Client Library Versions*

---

## Description

This function prints out the compiled and loaded client library versions.

## Usage

```
mysqlClientLibraryVersions()
```

## Value

A named integer vector of length two, the first element representing the compiled library version and the second element representint the loaded client library version.

---

`MySQLConnection-class` *Class MySQLConnection*

---

## Description

MySQLConnection class.

## Generators

The method `dbConnect` is the main generator.

## Extends

Class `"DBIConnection"`, directly. Class `"MySQLObject"`, directly. Class `"DBIObject"`, by class "DBIConnection". Class `"dbObjectId"`, by class "MySQLObject".

## Methods

**coerce** signature(from = "MySQLConnection", to = "MySQLResult"): ...

**dbCallProc** signature(conn = "MySQLConnection"): ...

**dbCommit** signature(conn = "MySQLConnection"): ...

**dbConnect** signature(drv = "MySQLConnection"): ...

**dbDisconnect** signature(conn = "MySQLConnection"): ...

**dbExistsTable** signature(conn = "MySQLConnection", name = "character"): ...

**dbGetException** signature(conn = "MySQLConnection"): ...

**dbGetInfo** signature(dbObj = "MySQLConnection"): ...

**dbGetQuery** signature(conn = "MySQLConnection", statement = "character"): ...

**dbListFields** signature(conn = "MySQLConnection", name = "character"): ...

**dbListResults** signature(conn = "MySQLConnection"): ...

**dbListTables** signature(conn = "MySQLConnection"): ...

**dbReadTable** signature(conn = "MySQLConnection", name = "character"): ...

**dbRemoveTable** signature(conn = "MySQLConnection", name = "character"): ...

**dbRollback** signature(conn = "MySQLConnection"): ...

**dbSendQuery** signature(conn = "MySQLConnection", statement = "character"): ...

**dbWriteTable** signature(conn = "MySQLConnection", name = "character", value = "data.frame"): ...

**summary** signature(object = "MySQLConnection"): ...

## References

See the Database Interface definition document DBI.pdf in the base directory of this package or http://developer.r-project.org/db.

## See Also

DBI base classes:

DBIObject-class DBIDriver-class DBIConnection-class DBIResult-class

MySQL classes:

MySQLObject-class MySQLDriver-class MySQLConnection-class MySQLResult-class

## Examples

```
## Not run:
drv <- dbDriver("MySQL)
con <- dbConnect(drv, dbname = "rsdbi.db")

## End(Not run)
```

mysqlDBApply *Apply R/S-Plus functions to remote groups of DBMS rows (experimental)*

## Description

Applies R/S-Plus functions to groups of remote DBMS rows without bringing an entire result set all at once. The result set is expected to be sorted by the grouping field.

## Usage

```
mysqlDBApply(res, INDEX, FUN = stop("must specify FUN"),
         begin = NULL,
         group.begin =  NULL,
         new.record = NULL,
         end = NULL,
         batchSize = 100, maxBatch = 1e6,
         ..., simplify = TRUE)
```

## Arguments

| | |
|---|---|
| res | a result set (see dbSendQuery). |
| INDEX | a character or integer specifying the field name or field number that defines the various groups. |
| FUN | a function to be invoked upon identifying the last row from every group. This function will be passed a data frame holding the records of the current group, a character string with the group label, plus any other arguments passed to dbApply as "...". |
| begin | a function of no arguments to be invoked just prior to retrieve the first row from the result set. |
| end | a function of no arguments to be invoked just after retrieving the last row from the result set. |
| group.begin | a function of one argument (the group label) to be invoked upon identifying a row from a new group. |
| new.record | a function to be invoked as each individual record is fetched. The first argument to this function is a one-row data.frame holding the new record. |
| batchSize | the default number of rows to bring from the remote result set. If needed, this is automatically extended to hold groups bigger than batchSize. |
| maxBatch | the absolute maximum of rows per group that may be extracted from the result set. |
| ... | any additional arguments to be passed to FUN. |
| simplify | Not yet implemented |

## Details

dbApply This function is meant to handle somewhat gracefully(?) large amounts of data from the DBMS by bringing into R manageable chunks (about batchSize records at a time, but not more than maxBatch); the idea is that the data from individual groups can be handled by R, but not all the groups at the same time.

The MySQL implementation mysqlDBApply allows us to register R functions that get invoked when certain fetching events occur. These include the "begin" event (no records have been yet fetched), "begin.group" (the record just fetched belongs to a new group), "new record" (every fetched record generates this event), "group.end" (the record just fetched was the last row of the current group), "end" (the very last record from the result set). Awk and perl programmers will find this paradigm very familiar (although SAP's ABAP language is closer to what we're doing).

## Value

A list with as many elements as there were groups in the result set.

## Note

This is an experimental version implemented only in R (there are plans, time permitting, to implement it in S-Plus).

The terminology that we're using is closer to SQL than R. In R what we're referring to "groups" are the individual levels of a factor (grouping field in our terminology).

## See Also

MySQL, dbSendQuery, fetch.

## Examples

```
## Not run:
## compute quanitiles for each network agent
con <- dbConnect(MySQL(), group="vitalAnalysis")
res <- dbSendQuery(con,
             "select Agent, ip_addr, DATA from pseudo_data order by Agent")
out <- dbApply(res, INDEX = "Agent",
        FUN = function(x, grp) quantile(x$DATA, names=FALSE))

## End(Not run)
```

---

# MySQLDriver-class *Class MySQLDriver*

---

## Description

An MySQL driver implementing the R/S-Plus database (DBI) API.

## Generators

The main generators are `dbDriver` and `MySQL`.

## Extends

Class `"DBIDriver"`, directly. Class `"MySQLObject"`, directly. Class `"DBIObject"`, by class "DBIDriver". Class `"dbObjectId"`, by class "MySQLObject".

## Methods

**coerce** `signature(from = "MySQLObject", to = "MySQLDriver")`: ...

**dbConnect** `signature(drv = "MySQLDriver")`: ...

**dbGetInfo** `signature(dbObj = "MySQLDriver")`: ...

**dbListConnections** `signature(drv = "MySQLDriver")`: ...

**dbUnloadDriver** `signature(drv = "MySQLDriver")`: ...

**summary** `signature(object = "MySQLDriver")`: ...

## References

See the Database Interface definition document `DBI.pdf` in the base directory of this package or `http://developer.r-project.org/db`.

## See Also

DBI base classes:

`DBIObject-class` `DBIDriver-class` `DBIConnection-class` `DBIResult-class`

MySQL classes:

`MySQLObject-class` `MySQLDriver-class` `MySQLConnection-class` `MySQLResult-class`

## Examples

```
## Not run:
drv <- dbDriver("MySQL")
con <- dbConnect(drv, "user/password@dbname")

## End(Not run)
```

MySQLObject-class *Class MySQLObject*

## Description

Base class for all MySQL-specific DBI classes

## Objects from the Class

A virtual Class: No objects may be created from it.

## Extends

Class "DBIObject", directly. Class "dbObjectId", directly.

## Methods

**coerce** signature(from = "MySQLObject", to = "MySQLriver"): ...

**dbDataType** signature(dbObj = "MySQLObject"): ...

**isSQLKeyword** signature(dbObj = "MySQLObject", name = "character"): ...

**make.db.names** signature(dbObj = "MySQLObject", snames = "character"): ...

**SQLKeywords** signature(dbObj = "MySQLObject"): ...

## References

See the Database Interface definition document DBI.pdf in the base directory of this package or http://developer.r-project.org/db.

## See Also

DBI base classes:

DBIObject-class DBIDriver-class DBIConnection-class DBIResult-class

MySQL classes:

MySQLObject-class MySQLDriver-class MySQLConnection-class MySQLResult-class

## Examples

```
## Not run:
drv <- dbDriver("MySQL")
con <- dbConnect(drv, dbname = "rsdbi.db")

## End(Not run)
```

MySQLResult-class *Class MySQLResult*

## Description

MySQL's query results class. This classes encapsulates the result of an SQL statement (either select or not).

## Generators

The main generator is dbSendQuery.

## Extends

Class "DBIResult", directly. Class "MySQLObject", directly. Class "DBIObject", by class "DBIResult". Class "dbObjectId", by class "MySQLObject".

## Methods

**coerce** signature(from = "MySQLConnection", to = "MySQLResult"): ...

**dbClearResult** signature(res = "MySQLResult"): ...

**dbColumnInfo** signature(res = "MySQLResult"): ...

**dbGetException** signature(conn = "MySQLResult"): ...

**dbGetInfo** signature(dbObj = "MySQLResult"): ...

**dbGetRowCount** signature(res = "MySQLResult"): ...

**dbGetRowsAffected** signature(res = "MySQLResult"): ...

**dbGetStatement** signature(res = "MySQLResult"): ...

**dbHasCompleted** signature(res = "MySQLResult"): ...

**dbListFields** signature(conn = "MySQLResult", name = "missing"): ...

**fetch** signature(res = "MySQLResult", n = "numeric"): ...

**fetch** signature(res = "MySQLResult", n = "missing"): ...

**summary** signature(object = "MySQLResult"): ...

## References

See the Database Interface definition document DBI.pdf in the base directory of this package or http://developer.r-project.org/db.

## See Also

DBI base classes:

DBIObject-class DBIDriver-class DBIConnection-class DBIResult-class

MySQL classes:

MySQLObject-class MySQLDriver-class MySQLConnection-class MySQLResult-class

## Examples

```
## Not run:
drv <- dbDriver("MySQL")
con <- dbConnect(drv, dbname = "rsdbi.db")

## End(Not run)
```

---

mysqlSupport *Support Functions*

---

## Description

These functions are the workhorse behind the RMySQL package, but users need not invoke these directly. For details see MySQL.

## Usage

```
## MySQLDriver-related
mysqlInitDriver(max.con=16, fetch.default.rec = 500, force.reload=FALSE)
mysqlDriverInfo(obj, what, ...)
mysqlDescribeDriver(obj, verbose = FALSE, ...)
mysqlCloseDriver(drv, ...)

## MySQLConnection-related
mysqlNewConnection(drv, dbname, username, password, host, unix.socket,
      port, client.flag, groups, default.file)
mysqlCloneConnection(con, ...)
mysqlConnectionInfo(obj, what, ...)
mysqlDescribeConnection(obj, verbose = FALSE, ...)
mysqlCloseConnection(con, ...)

## MySQLResult-related
mysqlExecStatement(con, statement)
mysqlFetch(res, n=0, ...)
mysqlQuickSQL(con, statement)
mysqlResultInfo(obj, what, ...)
mysqlDescribeResult(obj, verbose = FALSE, ...)
mysqlCloseResult(res, ...)
mysqlDescribeFields(res, ...)

## data mappings, convenience functions, and extensions
mysqlDataType(obj, ...)
mysqlReadTable(con, name, row.names = "row_names", check.names = TRUE, ...)
mysqlWriteTable(con, name, value, field.types, row.names = TRUE,
   overwrite=FALSE, append=FALSE, ..., allow.keywords = FALSE)
mysqlImportFile(con, name, value, field.types, overwrite=FALSE,
   append=FALSE, header, row.names, nrows=50, sep=",", eol="\n",
```

```
    skip = 0, quote='"', ...)
mysqlEscapeStrings(con, strings)
```

## Arguments

| | |
|---|---|
| max.con | positive integer specifying maximum number of open connections. The current default of 10 is hardcoded in the C code. |
| fetch.default.rec | default number of rows to fetch (move to R/S-Plus). This default is used in mysqlFetch. The default is 500. |
| force.reload | logical indicating whether to re-initialize the driver. This may be useful if you want to change the defaults (e.g., fetch.default.rec). Note that the driver is a singleton (subsequent inits just returned the previously initialized driver, thus this argument). |
| obj | any of the MySQL DBI objects (e.g., MySQLConnection, MySQLResult). |
| what | character vector of metadata to extract, e.g., "version", "statement", "isSelect". |
| verbose | logical controlling how much information to display. Defaults to FALSE. |
| drv | an MySQLDriver object as produced by mysqlInitDriver. |
| con | an MySQLConnection object as produced by mysqlNewConnection and mysqlCloneConnection. |
| res | an MySQLResult object as produced by by mysqlExecStatement. |
| username | a character string with the MySQL's user name. |
| password | character string with the MySQL's password. |
| groups | character vector with one or more MySQL group names. For details see MySQL. |
| default.file | filename of an alternate MySQL options file. |
| dbname | character string with the MySQL database name. |
| host | character string with the name (or IP address) of the machine hosting the database. Default is "", which is interpreted as localhost by the MySQL's API. |
| unix.socket | (optional) character string with a filename for the socket file name. Consult the MySQL documentation for details. |
| port | (optional) positive integer specifying the TCP port number that the MySQL server is listening to. Consult the MySQL documentation for details. |
| client.flag | (optional) integer setting flags for the client. Consult the MySQL documentation for details. |
| force | logical indicating whether to close a connection that has open result sets. The default is FALSE. |
| statement | character string holding one (and only one) SQL statement. |
| n | number of rows to fetch from the given result set. A value of -1 indicates to retrieve all the rows. The default of 0 specifies to extract whatever the fetch.default.rec was specified during driver initialization mysqlInit. |
| name | character vector of names (table names, fields, keywords). |
| value | a data.frame. |

| | |
|---|---|
| field.types | a list specifying the mapping from R/S-Plus fields in the data.frame value to SQL data types. The default is sapply(value, SQLDataType), see MySQLSQLType. |
| header | logical, does the input file have a header line? Default is the same heuristic used by read.table, i.e., TRUE if the first line has one fewer column that the second line. |
| row.names | a logical specifying whether to prepend the value data.frame row names or not. The default is TRUE. |
| check.names | a logical specifying whether to convert DBMS field names into legal S names. Default is TRUE. |
| overwrite | logical indicating whether to replace the table name with the contents of the data.frame value. The defauls is FALSE. |
| append | logical indicating whether to append value to the existing table name. |
| nrows | number of lines to rows to import using read.table from the input file to create the proper table definition. Default is 50. |
| sep | field separator character. |
| eol | end-of-line separator. |
| skip | number of lines to skip before reading data in the input file. |
| quote | the quote character used in the input file (defaults to \". |
| allow.keywords | logical indicating whether column names that happen to be MySQL keywords be used as column names in the resulting relation (table) being written. Defaults to FALSE, forcing mysqlWriteTable to modify column names to make them legal MySQL identifiers. |
| strings | a character vector of strings to be escaped |
| ... | placeholder for future use. |

## Value

mysqlInitDriver returns an MySQLDriver object.

mysqlDriverInfo returns a list of name-value metadata pairs.

mysqlDescribeDriver returns NULL (displays the object's metadata).

mysqlCloseDriver returns a logical indicating whether the operation succeeded or not.

mysqlNewConnection returns an MySQLConnection object.

mysqlCloneConnection returns an MySQLConnection object.

mysqlConnectionInforeturns a list of name-value metadata pairs.

mysqlDescribeConnection returns NULL (displays the object's metadata).

mysqlCloseConnection returns a logical indicating whether the operation succeeded or not.

mysqlExecStatement returns an MySQLResult object.

mysqlFetch returns a data.frame.

mysqlQuickSQL returns either a data.frame if the statement is a select-like or NULL otherwise.

mysqlDescribeResult returns NULL (displays the object's metadata).

mysqlCloseResult returns a logical indicating whether the operation succeeded or not.

mysqlDescribeFields returns a data.frame with one row per field with columns name, Sclass, type, len, precision, scale, and nullOK which fully describe each field in a result set. Except for Sclass (which shows the mapping of the field type into an R/S-Plus class) all the information pertains to MySQL's data storage attributes.

mysqlReadTable returns a data.frame with the contents of the DBMS table.

mysqlWriteTable returns a logical indicating whether the operation succeeded or not.

mysqlDataType retuns a character string with the closest

mysqlResultInfo returns a list of name-value metadata pairs.

mysqlEscapeStrings returns a character vector with each string escaped for MySQL special characters (such as single and double quotes). This is done using the character set used by the connection con.

## Constants

.MySQLPkgName (currently "RMySQL"), .MySQLPkgVersion (the R package version), .MySQLPkgRCS (the RCS revision), .MySQL.NA.string (character that MySQL uses to denote NULL on input), .MySQLSQLKeywords (a lot!) .conflicts.OK.

---

summary-methods *Summarize an MySQL object*

---

## Description

These methods are straight-forward implementations of the corresponding generic functions.

## Methods

**object = "DBIObject"** Provides relevant metadata information on object, for instance, the MySQL server file, the SQL statement associated with a result set, etc.

**from** object to be coerced

**to** coercion class

**x** object to format or print or show

# Index