

The RMySQL Package

October 27, 2006

Version 0.5-10

Title R interface to the MySQL database

Author David A. James and Saikat DebRoy

Maintainer David A. James <daj025@gmail.com>

Description Database interface and MySQL driver for R. This version complies with the database interface definition as implemented in the package DBI 0.1-8.

SaveImage yes

Depends R (>= 2.3.0), methods, DBI (>= 0.1-11)

License GPL

URL stat.bell-labs.com/RS-DBI www.mysql.com www.omegahat.org

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

R topics documented:

MySQL	2
MySQLConnection-class	4
MySQLDriver-class	6
MySQLObject-class	7
MySQLResult-class	8
RMySQL-package	9
S4R	12
dbApply-methods	12
dbApply	13
dbBuildTableDefinition	14
dbCallProc-methods	15
dbCommit-methods	15
dbConnect-methods	16
dbDataType-methods	17
dbDriver-methods	18
dbGetInfo-methods	19
dbListTables-methods	20
dbObjectId-class	20
dbReadTable-methods	21

dbSendQuery-methods	23
dbSetDataMappings-methods	24
fetch-methods	24
isIdCurrent	25
make.db.names-methods	26
mysqlDBApply	27
mysqlSupport	29
safe.write	32
summary-methods	33

Index 34

MySQL *Instantiate a MySQL client from the current R or S-Plus 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

<code>max.con</code>	maximum number of connections that are intended to have open at one time. There's no intrinsic limit, since strictly speaking this limit applies to MySQL <i>servers</i> , but clients can have (at least in theory) more than this. Typically there are at most a handful of open connections, thus the internal <code>RMySQL</code> code uses a very simple linear search algorithm to manage its connection table.
<code>fetch.default.rec</code>	number of records to fetch at one time from the database. (The <code>fetch</code> method uses this number as a default.)
<code>force.reload</code>	should the client code be reloaded (reinitialize)? Setting this to <code>TRUE</code> 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/S-Plus 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 readable only by you. Inside each section, MySQL parameters may be specified one per line (e.g., `user = opto`). The R/S-Plus MySQL implementation always parses the `[client]` and `[rs-dbi]` sections, but you may define your own project-specific sections to tailor its environment; if the same parameter appears more than once, the last (closer to the bottom) occurrence is used.

If you define a section, 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

The `dbname` cannot go in the `[client]` section, but you may safely include it under the `[rs-dbi]` section or one you define yourself.

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](#)
[dbGetAffectedRows](#)

Examples

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

# open the connection using user, password, 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)
```

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)

```

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 = "MySQLDriver"):...

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 class 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)
```

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. 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)
```

7. 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 your 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="router_shield").

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:

[DBI 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 dbGetAffectedRows](#)

Examples

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

# open the connection using user, password, 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)
```

S4R

*R compatibility with S version 4/Splus5+ support functions***Description**

These objects ease the task of porting functions into R from S Version 4 and S-Plus 5.0 and later. See the documentation there. May be obsolete in the future.

Usage

```
usingR(major, minor)
```

Examples

```
## Not run:
rc <- try(fetch(res, n = -1))
if(inherit(rc, ErrorClass))
  stop("could not fetch the data")
## 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 quantiles 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	<i>Apply R/S-Plus functions to remote groups of DBMS rows (experimental)</i>
---------	--

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

```
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 quantiles 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 coercible 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.

password string with the MySQL password.

dbname string with the database name.

host string identifying the host machine running the MySQL server an empty string `" "` is interpreted as `"localhost"`.

unix.socket (optional) unix socket or named pipe.

port (optional) 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 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, password, 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.

Methods

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 simultaneous 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)
```

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)
```

 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, its value is determined following `read.table` convention, namely, it is set to `TRUE` if and only if the first row has one fewer field than 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`).

names 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 overridden 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)
```

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

<code>res</code>	a result set (see dbSendQuery).
<code>INDEX</code>	a character or integer specifying the field name or field number that defines the various groups.
<code>FUN</code>	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 <code>dbApply</code> as "...".
<code>begin</code>	a function of no arguments to be invoked just prior to retrieve the first row from the result set.
<code>end</code>	a function of no arguments to be invoked just after retrieving the last row from the result set.
<code>group.begin</code>	a function of one argument (the group label) to be invoked upon identifying a row from a new group
<code>new.record</code>	a function to be invoked as each individual record is fetched. The first argument to this function is a one-row <code>data.frame</code> holding the new record.
<code>batchSize</code>	the default number of rows to bring from the remote result set. If needed, this is automatically extended to hold groups bigger than <code>batchSize</code> .
<code>maxBatch</code>	the absolute maximum of rows per group that may be extracted from the result set.
<code>...</code>	any additional arguments to be passed to <code>FUN</code> .
<code>simplify</code>	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 quantiles 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)
```

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="'', ...)
```

Arguments

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

<code>row.names</code>	a logical specifying whether to prepend the <code>value</code> <code>data.frame</code> row names or not. The default is <code>TRUE</code> .
<code>check.names</code>	a logical specifying whether to convert DBMS field names into legal S names. Default is <code>TRUE</code> .
<code>overwrite</code>	logical indicating whether to replace the table name with the contents of the <code>data.frame</code> <code>value</code> . The default is <code>FALSE</code> .
<code>append</code>	logical indicating whether to append <code>value</code> to the existing table name.
<code>nrows</code>	number of lines to rows to import using <code>read.table</code> from the input file to create the proper table definition. Default is 50.
<code>sep</code>	field separator character.
<code>eol</code>	end-of-line separator.
<code>skip</code>	number of lines to skip before reading data in the input file.
<code>quote</code>	the quote character used in the input file (defaults to <code>"</code>).
<code>allow.keywords</code>	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 <code>FALSE</code> , forcing <code>mysqlWriteTable</code> to modify column names to make them legal MySQL identifiers.
<code>...</code>	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.

`mysqlConnectionInfo` returns 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` returns a character string with the closest

`mysqlResultInfo` returns a list of name-value metadata pairs.

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`.

<code>safe.write</code>	<i>Write a data.frame avoiding exceeding memory limits</i>
-------------------------	--

Description

This function batches calls to `write.table` to avoid exceeding memory limits for very large `data.frames`.

Usage

```
safe.write(value, file, batch, ...)
```

Arguments

<code>value</code>	a <code>data.frame</code> ;
<code>file</code>	a file object (connection, file name, etc).
<code>batch</code>	maximum number of rows to write at a time.
<code>...</code>	any other arguments are passed to <code>write.table</code> .

Details

The function has a while loop invoking `write.table` for subsets of `batch` rows of `value`. Since this is a helper function for `mysqlWriteTable`, it has hardcoded other arguments to `write.table`.

Value

NULL, invisibly.

Note

No error checking whatsoever is done.

See Also

[write.table](#)

Examples

```
## Not run:
  ctr.file <- file("dump.sqlloader", "w")
  safe.write(big.data, file = ctr.file, batch = 25000)
## End(Not run)
```

summary-methods	<i>Summarize an MySQL object</i>
-----------------	----------------------------------

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

*Topic **classes**

- dbObjectId-class, 19
- MySQLConnection-class, 4
- MySQLDriver-class, 5
- MySQLObject-class, 6
- MySQLResult-class, 7

*Topic **database**

- dbApply, 12
- dbApply-methods, 11
- dbBuildTableDefinition, 13
- dbCallProc-methods, 14
- dbCommit-methods, 14
- dbConnect-methods, 15
- dbDataType-methods, 16
- dbDriver-methods, 17
- dbGetInfo-methods, 18
- dbListTables-methods, 19
- dbObjectId-class, 19
- dbReadTable-methods, 20
- dbSendQuery-methods, 22
- dbSetDataMappings-methods, 23
- fetch-methods, 23
- isIdCurrent, 24
- make.db.names-methods, 25
- MySQL, 1
- MySQLConnection-class, 4
- mysqlDBApply, 26
- MySQLDriver-class, 5
- MySQLObject-class, 6
- MySQLResult-class, 7
- mysqlSupport, 28
- RMySQL-package, 8
- summary-methods, 32

*Topic **datasets**

- mysqlSupport, 28

*Topic **interface**

- dbApply, 12
- dbApply-methods, 11
- dbBuildTableDefinition, 13
- dbCallProc-methods, 14
- dbCommit-methods, 14
- dbConnect-methods, 15
- dbDataType-methods, 16

- dbDriver-methods, 17
- dbGetInfo-methods, 18
- dbListTables-methods, 19
- dbObjectId-class, 19
- dbReadTable-methods, 20
- dbSendQuery-methods, 22
- dbSetDataMappings-methods, 23
- fetch-methods, 23
- isIdCurrent, 24
- make.db.names-methods, 25
- MySQL, 1
- MySQLConnection-class, 4
- mysqlDBApply, 26
- MySQLDriver-class, 5
- MySQLObject-class, 6
- MySQLResult-class, 7
- mysqlSupport, 28
- RMySQL-package, 8
- summary-methods, 32

*Topic **internal**

- S4R, 11
- safe.write, 31

*Topic **methods**

- dbBuildTableDefinition, 13
- dbCallProc-methods, 14
- dbCommit-methods, 14
- dbConnect-methods, 15
- dbDataType-methods, 16
- dbDriver-methods, 17
- dbGetInfo-methods, 18
- dbListTables-methods, 19
- dbReadTable-methods, 20
- dbSendQuery-methods, 22
- dbSetDataMappings-methods, 23
- fetch-methods, 23
- make.db.names-methods, 25
- summary-methods, 32

*Topic **package**

- RMySQL-package, 8

*Topic **programming**

- dbApply, 12
- dbApply-methods, 11
- mysqlDBApply, 26

- `.MySQL.NA.string` (*mysqlSupport*), 28
- `.MySQLPkgName` (*mysqlSupport*), 28
- `.MySQLPkgRCS` (*mysqlSupport*), 28
- `.MySQLPkgVersion` (*mysqlSupport*), 28
- `.MySQLSQLKeywords` (*mysqlSupport*), 28
- `.conflicts.OK` (*mysqlSupport*), 28
- `coerce`, 4, 6, 7, 20
- `coerce, dbObjectId, character-method` (*summary-methods*), 32
- `coerce, dbObjectId, integer-method` (*summary-methods*), 32
- `coerce, dbObjectId, numeric-method` (*summary-methods*), 32
- `coerce, MySQLConnection, MySQLDriver-method` (*summary-methods*), 32
- `coerce, MySQLConnection, MySQLResult-method` (*summary-methods*), 32
- `coerce, MySQLObject, MySQLDriver-method` (*summary-methods*), 32
- `coerce, MySQLResult, MySQLConnection-method` (*summary-methods*), 32
- `coerce, MySQLResult, MySQLDriver-method` (*summary-methods*), 32
- `coerce-methods` (*summary-methods*), 32
- `dbApply`, 12
- `dbApply, MySQLResult-method` (*dbApply-methods*), 11
- `dbApply-methods`, 11
- `dbBuildTableDefinition`, 13
- `dbCallProc`, 4
- `dbCallProc, MySQLConnection-method` (*dbCallProc-methods*), 14
- `dbCallProc-methods`, 14
- `dbClearResult`, 3, 7, 10, 24
- `dbClearResult, MySQLResult-method` (*dbSendQuery-methods*), 22
- `dbClearResult-methods` (*dbSendQuery-methods*), 22
- `dbColumnInfo`, 3, 7, 10, 19, 23
- `dbColumnInfo, MySQLResult-method` (*dbGetInfo-methods*), 18
- `dbColumnInfo-methods` (*dbGetInfo-methods*), 18
- `dbCommit`, 3, 4, 10, 13, 14, 16–18, 21, 22, 24
- `dbCommit, MySQLConnection-method` (*dbCommit-methods*), 14
- `dbCommit-methods`, 14
- `dbConnect`, 2–4, 6, 10, 13, 14, 16–19, 21, 22, 24, 25
- `dbConnect, character-method` (*dbConnect-methods*), 15
- `dbConnect, MySQLConnection-method` (*dbConnect-methods*), 15
- `dbConnect, MySQLDriver-method` (*dbConnect-methods*), 15
- `dbConnect-methods`, 15
- `dbDataType`, 6, 21
- `dbDataType, MySQLObject-method` (*dbDataType-methods*), 16
- `dbDataType-methods`, 16
- `dbDisconnect`, 3, 4, 10
- `dbDisconnect, MySQLConnection-method` (*dbConnect-methods*), 15
- `dbDisconnect-methods` (*dbConnect-methods*), 15
- `dbDriver`, 3, 5, 10, 18, 19, 21, 22, 25
- `dbDriver, character-method` (*dbDriver-methods*), 17
- `dbDriver-methods`, 17
- `dbExistsTable`, 4, 26
- `dbExistsTable, MySQLConnection, character-method` (*dbReadTable-methods*), 20
- `dbExistsTable-methods` (*dbReadTable-methods*), 20
- `dbGetAffectedRows`, 3, 10
- `dbGetDBIVersion`, 3, 10
- `dbGetDBIVersion-methods` (*dbGetInfo-methods*), 18
- `dbGetException`, 3, 4, 7, 10
- `dbGetException, MySQLConnection-method` (*dbSendQuery-methods*), 22
- `dbGetException, MySQLResult-method` (*dbSendQuery-methods*), 22
- `dbGetException-methods` (*dbSendQuery-methods*), 22
- `dbGetInfo`, 3, 4, 6, 7, 10, 13, 14, 16–19, 21, 22, 24
- `dbGetInfo` (*dbGetInfo-methods*), 18
- `dbGetInfo, MySQLConnection-method` (*dbGetInfo-methods*), 18
- `dbGetInfo, MySQLDriver-method` (*dbGetInfo-methods*), 18
- `dbGetInfo, MySQLObject-method` (*dbGetInfo-methods*), 18
- `dbGetInfo, MySQLResult-method` (*dbGetInfo-methods*), 18
- `dbGetInfo-methods`, 18
- `dbGetQuery`, 3, 4, 10, 13, 14, 16–18, 21, 24
- `dbGetQuery, MySQLConnection, character-method`

- (dbSendQuery-methods)*, 22
- dbGetQuery-methods
 - (dbSendQuery-methods)*, 22
- dbGetRowCount, 3, 7, 10
- dbGetRowCount, MySQLResult-method
 - (dbGetInfo-methods)*, 18
- dbGetRowCount-methods
 - (dbGetInfo-methods)*, 18
- dbGetRowsAffected, 7
- dbGetRowsAffected, MySQLResult-method
 - (dbGetInfo-methods)*, 18
- dbGetRowsAffected-methods
 - (dbGetInfo-methods)*, 18
- dbGetStatement, 3, 7, 10
- dbGetStatement, MySQLResult-method
 - (dbGetInfo-methods)*, 18
- dbGetStatement-methods
 - (dbGetInfo-methods)*, 18
- dbHasCompleted, 3, 7, 10
- dbHasCompleted, MySQLResult-method
 - (dbGetInfo-methods)*, 18
- dbHasCompleted-methods
 - (dbGetInfo-methods)*, 18
- DBI, 10
- DBIConnection-class, 5–8
- DBIDriver-class, 5–8
- DBIObject-class, 5–8
- DBIResult-class, 5–8
- dbListConnections, 3, 6, 10
- dbListConnections, MySQLDriver-method
 - (dbListTables-methods)*, 19
- dbListConnections-methods
 - (dbListTables-methods)*, 19
- dbListFields, 4, 7
- dbListFields, MySQLConnection, character-method
 - (dbListTables-methods)*, 19
- dbListFields, MySQLResult, missing-method
 - (dbListTables-methods)*, 19
- dbListFields-methods
 - (dbListTables-methods)*, 19
- dbListResults, 3, 4, 10
- dbListResults, MySQLConnection-method
 - (dbListTables-methods)*, 19
- dbListResults-methods
 - (dbListTables-methods)*, 19
- dbListTables, 3, 4, 10, 17, 18, 21, 26
- dbListTables, MySQLConnection-method
 - (dbListTables-methods)*, 19
- dbListTables-methods, 19
- DBObjectId-class, 19
- dbReadTable, 4, 13, 14, 16–18, 21, 22, 24, 26
 - dbReadTable, MySQLConnection, character-method
 - (dbReadTable-methods)*, 20
 - dbReadTable-methods, 20
 - dbRemoveTable, 5, 26
 - dbRemoveTable, MySQLConnection, character-method
 - (dbReadTable-methods)*, 20
 - dbRemoveTable-methods
 - (dbReadTable-methods)*, 20
 - dbRollback, 3, 5, 10
 - dbRollback, MySQLConnection-method
 - (dbCommit-methods)*, 14
 - dbRollback-methods
 - (dbCommit-methods)*, 14
 - dbSendQuery, 3, 5, 7, 10–14, 16–19, 21, 23–25, 27
 - dbSendQuery, MySQLConnection, character-method
 - (dbSendQuery-methods)*, 22
 - dbSendQuery-methods, 22
 - dbSetDataMappings, MySQLResult, data.frame-method
 - (dbSetDataMappings-methods)*, 23
 - dbSetDataMappings-methods, 23
 - dbUnloadDriver, 3, 6, 10
 - dbUnloadDriver, MySQLDriver-method
 - (dbDriver-methods)*, 17
 - dbUnloadDriver-methods
 - (dbDriver-methods)*, 17
 - dbWriteTable, 5, 26
 - dbWriteTable, MySQLConnection, character, character-method
 - (dbReadTable-methods)*, 20
 - dbWriteTable, MySQLConnection, character, data.frame-method
 - (dbReadTable-methods)*, 20
 - dbWriteTable-methods
 - (dbReadTable-methods)*, 20
- ErrorClass (S4R), 11
- fetch, 2, 3, 8, 10, 12–14, 16–18, 21–23, 25, 27
- fetch, MySQLResult, missing-method
 - (fetch-methods)*, 23
- fetch, MySQLResult, numeric-method
 - (fetch-methods)*, 23
- fetch-methods, 23
- format, 20
- format, dbObjectId-method
 - (summary-methods)*, 32
- format-methods (*summary-methods*), 32
- isIdCurrent, 24
- isSQLKeyword, 6, 16, 21

- isSQLKeyword, MySQLObject, character-method, [20](#)
- (*make.db.names-methods*), [25](#)
- isSQLKeyword-methods
- (*make.db.names-methods*), [25](#)
- make.db.names, [6](#), [16](#), [20](#), [21](#)
- make.db.names, MySQLObject, character-method, [25](#)
- (*make.db.names-methods*), [25](#)
- make.db.names-methods, [25](#)
- MySQL, [1](#), [5](#), [12–19](#), [21–24](#), [26–29](#)
- mysqlCloneConnection
- (*mysqlSupport*), [28](#)
- mysqlCloseConnection
- (*mysqlSupport*), [28](#)
- mysqlCloseDriver (*mysqlSupport*), [28](#)
- mysqlCloseResult (*mysqlSupport*), [28](#)
- MySQLConnection-class, [5–8](#)
- MySQLConnection-class, [4](#)
- mysqlConnectionInfo
- (*mysqlSupport*), [28](#)
- mysqlDataType (*mysqlSupport*), [28](#)
- mysqlDBApply, [12](#), [26](#)
- mysqlDescribeConnection
- (*mysqlSupport*), [28](#)
- mysqlDescribeDriver
- (*mysqlSupport*), [28](#)
- mysqlDescribeFields
- (*mysqlSupport*), [28](#)
- mysqlDescribeResult
- (*mysqlSupport*), [28](#)
- MySQLDriver-class, [5–8](#)
- MySQLDriver-class, [5](#)
- mysqlDriverInfo (*mysqlSupport*), [28](#)
- mysqlExecStatement
- (*mysqlSupport*), [28](#)
- mysqlFetch (*mysqlSupport*), [28](#)
- mysqlImportFile, [21](#)
- mysqlImportFile (*mysqlSupport*), [28](#)
- mysqlInitDriver (*mysqlSupport*), [28](#)
- mysqlNewConnection
- (*mysqlSupport*), [28](#)
- MySQLObject-class, [5–8](#)
- MySQLObject-class, [6](#)
- mysqlQuickSQL (*mysqlSupport*), [28](#)
- mysqlReadTable (*mysqlSupport*), [28](#)
- MySQLResult-class, [5–8](#)
- MySQLResult-class, [7](#)
- mysqlResultInfo (*mysqlSupport*), [28](#)
- mysqlSupport, [28](#)
- mysqlWriteTable, [31](#)
- mysqlWriteTable (*mysqlSupport*), [28](#)
- print, [20](#)
- print, dbObjectId-method
- (*summary-methods*), [32](#)
- read.table, [21](#)
- RMySQL (*RMySQL-package*), [8](#)
- RMySQL-package, [8](#)
- S4R, [11](#)
- safe.write, [31](#)
- show, [20](#)
- show, dbObjectId-method
- (*summary-methods*), [32](#)
- show-methods (*summary-methods*), [32](#)
- SQLKeywords, [6](#)
- SQLKeywords, missing-method
- (*make.db.names-methods*), [25](#)
- SQLKeywords, MySQLObject-method
- (*make.db.names-methods*), [25](#)
- SQLKeywords-methods
- (*make.db.names-methods*), [25](#)
- summary, [3](#), [6](#), [8](#), [10](#)
- summary, MySQLConnection-method
- (*summary-methods*), [32](#)
- summary, MySQLDriver-method
- (*summary-methods*), [32](#)
- summary, MySQLObject-method
- (*summary-methods*), [32](#)
- summary, MySQLResult-method
- (*summary-methods*), [32](#)
- summary-methods, [32](#)
- usingR (*S4R*), [11](#)
- write.table, [31](#)