

# Package ‘RMySQL’

March 29, 2017

**Version** 0.10.11

**Title** Database Interface and 'MySQL' Driver for R

**Description** A 'DBI' interface to 'MySQL' / 'MariaDB'. The CRAN version of this package contains an old branch based on legacy code from S-PLUS, which being phased out. A modern rewrite based on 'Rcpp' can be obtained from the 'Github' repository.

**Depends** R (>= 2.8.0), DBI (>= 0.4)

**Imports** methods

**License** GPL-2

**URL** <https://github.com/rstats-db/RMySQL#readme> (devel)  
<https://downloads.mariadb.org/connector-c/> (upstream)

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

**SystemRequirements** libmariadb-client-dev | libmariadb-client-lgpl-dev  
| libmysqlclient-dev (deb), mariadb-devel (rpm), mariadb |  
mysql-connector-c (brew), mysql56\_dev (csw)

**NeedsCompilation** yes

**Collate** 'mysql.R' 'driver.R' 'connection.R' 'data-type.R' 'default.R'  
'escaping.R' 'result.R' 'extension.R' 'is-valid.R' 'table.R'  
'transaction.R'

**Suggests** testthat

**Author** Jeroen Ooms [aut, cre],  
David James [aut],  
Saikat DebRoy [aut],  
Hadley Wickham [aut],  
Jeffrey Horner [aut],  
RStudio [cph]

**Maintainer** Jeroen Ooms <jeroen@berkeley.edu>

**Repository** CRAN

**Date/Publication** 2017-03-29 10:15:02 UTC

## R topics documented:

constants	2
db-meta	2
dbApply	3
dbConnect,MySQLDriver-method	5
dbDataType,MySQLDriver-method	7
dbEscapeStrings	7
dbFetch,MySQLResult,numeric-method	8
dbGetInfo,MySQLDriver-method	10
dbNextResult	11
dbReadTable,MySQLConnection,character-method	12
dbUnloadDriver,MySQLDriver-method	13
dbWriteTable,MySQLConnection,character,data.frame-method	14
isIdCurrent	15
make.db.names,MySQLConnection,character-method	16
mysqlClientLibraryVersions	17
MySQLDriver-class	17
mysqlHasDefault	18
result-meta	19
transactions	20
<b>Index</b>	<b>21</b>

---

constants	<i>Constants</i>
-----------	------------------

---

### Description

Constants

### Constants

.MySQLPkgName (currently "RMySQL"), .MySQLPkgVersion (the R package version), .MySQLPkgRCS (the RCS revision), .MySQLSQLKeywords (a lot!)

---

db-meta	<i>Database interface meta-data</i>
---------	-------------------------------------

---

### Description

Database interface meta-data

**Usage**

```
## S4 method for signature 'MySQLConnection'
dbGetInfo(dbObj, what = "", ...)

## S4 method for signature 'MySQLConnection'
dbListResults(conn, ...)

## S4 method for signature 'MySQLConnection'
summary(object, verbose = FALSE, ...)

## S4 method for signature 'MySQLConnection'
dbGetException(conn, ...)

## S4 method for signature 'MySQLConnection'
show(object)
```

**Arguments**

what	optional
...	Other arguments for compatibility with generic.
conn, dbObj, object	MySQLConnection object.
verbose	If TRUE, add extra info.

**Examples**

```
if (mysqlHasDefault()) {
  con <- dbConnect(RMySQL::MySQL(), dbname = "test")

  summary(con)

  dbGetInfo(con)
  dbListResults(con)
  dbListTables(con)

  dbDisconnect(con)
}
```

**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.

The MySQL implementation 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).

**Usage**

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

```
## S4 method for signature 'MySQLResult'
dbApply(res, INDEX, FUN = stop("must specify FUN"),
        begin = NULL, group.begin = NULL, new.record = NULL, end = NULL,
        batchSize = 100, maxBatch = 1e+06, ..., simplify = TRUE)
```

**Arguments**

res	a result set (see <a href="#">dbSendQuery</a> ).
...	any additional arguments to be passed to FUN.
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.
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.
end	a function of no arguments to be invoked just after retrieving the last row from the result set.
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.
simplify	Not yet implemented

**Details**

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.

**Value**

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

**Examples**

```
if (mysqlHasDefault()) {
  con <- dbConnect(RMySQL::MySQL(), dbname = "test")

  dbWriteTable(con, "mtcars", mtcars, overwrite = TRUE)
  res <- dbSendQuery(con, "SELECT * FROM mtcars ORDER BY cyl")
  dbApply(res, "cyl", function(x, grp) quantile(x$mpg, names=FALSE))

  dbClearResult(res)
  dbRemoveTable(con, "mtcars")
  dbDisconnect(con)
}
```

---

 dbConnect,MySQLDriver-method

*Connect/disconnect to a MySQL DBMS*

---

**Description**

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

**Usage**

```
## S4 method for signature 'MySQLDriver'
dbConnect(drv, dbname = NULL, username = NULL,
  password = NULL, host = NULL, unix.socket = NULL, port = 0,
  client.flag = 0, groups = "rs-dbi", default.file = NULL, ...)

## S4 method for signature 'MySQLConnection'
dbConnect(drv, ...)

## S4 method for signature 'MySQLConnection'
dbDisconnect(conn, ...)
```

**Arguments**

<code>drv</code>	an object of class <code>MySQLDriver</code> , or the character string "MySQL" or an <code>MySQLConnection</code> .
<code>dbname</code>	string with the database name or NULL. If not NULL, the connection sets the default daabase to this value.
<code>username,password</code>	Username and password. If username omitted, defaults to the current user. If password is ommitted, only users without a password can log in.
<code>host</code>	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.
<code>unix.socket</code>	(optional) string of the unix socket or named pipe.
<code>port</code>	(optional) integer of the TCP/IP default port.
<code>client.flag</code>	(optional) integer setting various MySQL client flags. See the MySQL manual for details.
<code>groups</code>	string identifying a section in the <code>default.file</code> to use for setting authentication parameters (see <a href="#">MySQL</a> ).
<code>default.file</code>	string of the filename with MySQL client options. Defaults to <code>\$HOME/.my.cnf</code>
<code>...</code>	Unused, needed for compatibility with generic.
<code>conn</code>	an <code>MySQLConnection</code> object as produced by <code>dbConnect</code> .

**Examples**

```
## Not run:
# Connect to a MySQL database running locally
con <- dbConnect(RMySQL::MySQL(), dbname = "mydb")
# Connect to a remote database with username and password
con <- dbConnect(RMySQL::MySQL(), host = "mydb.mycompany.com",
  user = "abc", password = "def")
# But instead of supplying the username and password in code, it's usually
# better to set up a group in your .my.cnf (usually located in your home
# directory). Then it's less likely you'll inadvertently share them.
con <- dbConnect(RMySQL::MySQL(), group = "test")

# Always cleanup by disconnecting the database
dbDisconnect(con)

## End(Not run)

# All examples use the rs-dbi group by default.
if (mysqlHasDefault()) {
  con <- dbConnect(RMySQL::MySQL(), dbname = "test")
  summary(con)
  dbDisconnect(con)
}
```

---

 dbDataType,MySQLDriver-method

*Determine the SQL Data Type of an S object*


---

### Description

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

### Usage

```
## S4 method for signature 'MySQLDriver'
dbDataType(dbObj, obj)
```

```
## S4 method for signature 'MySQLConnection'
dbDataType(dbObj, obj)
```

### Arguments

dbObj	A MySQLDriver or MySQLConnection.
obj	R/S-Plus object whose SQL type we want to determine.
...	any other parameters that individual methods may need.

### Examples

```
dbDataType(RMySQL::MySQL(), "a")
dbDataType(RMySQL::MySQL(), 1:3)
dbDataType(RMySQL::MySQL(), 2.5)
```

---

 dbEscapeStrings      *Escape SQL-special characters in strings.*


---

### Description

Escape SQL-special characters in strings.

### Usage

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

```
## S4 method for signature 'MySQLConnection,character'
dbEscapeStrings(con, strings)
```

```
## S4 method for signature 'MySQLResult,character'
dbEscapeStrings(con, strings, ...)
```

**Arguments**

con                    a connection object (see [dbConnect](#)).

strings                a character vector.

...                    any additional arguments to be passed to the dispatched method.

**Value**

A character vector with SQL special characters properly escaped.

**Examples**

```
if (mysqlHasDefault()) {
con <- dbConnect(RMySQL::MySQL(), dbname = "test")

tmp <- sprintf("SELECT * FROM emp WHERE lname = %s", "O'Reilly")
dbEscapeStrings(con, tmp)

dbDisconnect(con)
}
```

---

dbFetch,MySQLResult,numeric-method

*Execute a SQL statement on a database connection.*

---

**Description**

To retrieve results a chunk at a time, use `dbSendQuery`, `dbFetch`, then `dbClearResult`. Alternatively, if you want all the results (and they'll fit in memory) use `dbGetQuery` which sends, fetches and clears for you.

**Usage**

```
## S4 method for signature 'MySQLResult,numeric'
dbFetch(res, n = -1, ...)

## S4 method for signature 'MySQLResult,numeric'
fetch(res, n = -1, ...)

## S4 method for signature 'MySQLResult,missing'
dbFetch(res, n = -1, ...)

## S4 method for signature 'MySQLResult,missing'
fetch(res, n = -1, ...)

## S4 method for signature 'MySQLConnection,character'
dbSendQuery(conn, statement)
```



```
## S4 method for signature 'MySQLResult'
dbClearResult(res, ...)

## S4 method for signature 'MySQLResult'
dbGetInfo(dbObj, what = "", ...)

## S4 method for signature 'MySQLResult'
dbGetStatement(res, ...)

## S4 method for signature 'MySQLResult,missing'
dbListFields(conn, name, ...)
```

### Arguments

res,dbObj	A <a href="#">MySQLResult</a> object.
n	maximum number of records to retrieve per fetch. Use -1 to retrieve all pending records; use 0 for to fetch the default number of rows as defined in <a href="#">MySQL</a>
...	Unused. Needed for compatibility with generic.
conn	an <a href="#">MySQLConnection</a> object.
statement	a character vector of length one specifying the SQL statement that should be executed. Only a single SQL statement should be provided.
what	optional
name	Table name.

### Details

fetch() will be deprecated in the near future; please use dbFetch() instead.

### Examples

```
if (mysqlHasDefault()) {
  con <- dbConnect(RMySQL::MySQL(), dbname = "test")
  dbWriteTable(con, "arrests", datasets::USArrests, overwrite = TRUE)

  # Run query to get results as dataframe
  dbGetQuery(con, "SELECT * FROM arrests limit 3")

  # Send query to pull requests in batches
  res <- dbSendQuery(con, "SELECT * FROM arrests")
  data <- dbFetch(res, n = 2)
  data
  dbHasCompleted(res)

  dbListResults(con)
  dbClearResult(res)
  dbRemoveTable(con, "arrests")
  dbDisconnect(con)
}
```

---

dbGetInfo,MySQLDriver-method

*Get information about a MySQL driver.*

---

## Description

Get information about a MySQL driver.

## Usage

```
## S4 method for signature 'MySQLDriver'  
dbGetInfo(dbObj, what = "", ...)
```

```
## S4 method for signature 'MySQLDriver'  
dbListConnections(drv, ...)
```

```
## S4 method for signature 'MySQLDriver'  
summary(object, verbose = FALSE, ...)
```

```
## S4 method for signature 'MySQLDriver'  
show(object)
```

## Arguments

dbObj,object,drv	Object created by <a href="#">MySQL</a> .
what	Optional
...	Ignored. Needed for compatibility with generic.
verbose	If TRUE, print extra info.

## Examples

```
db <- RMySQL::MySQL()  
  
db  
dbGetInfo(db)  
dbListConnections(db)  
summary(db)
```

---

dbNextResult	<i>Fetch next result set from an SQL script or stored procedure (experimental)</i>
--------------	------------------------------------------------------------------------------------

---

### Description

SQL scripts (i.e., multiple SQL statements separated by ';') and stored procedures oftentimes generate multiple result sets. These 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.

### Usage

```
dbNextResult(con, ...)

## S4 method for signature 'MySQLConnection'
dbNextResult(con, ...)

dbMoreResults(con, ...)

## S4 method for signature 'MySQLConnection'
dbMoreResults(con, ...)
```

### Arguments

con	a connection object (see <a href="#">dbConnect</a> ).
...	any additional arguments to be passed to the dispatched method

### 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.

### Examples

```
if (mysqlHasDefault()) {
  con <- dbConnect(RMySQL::MySQL(), dbname = "test", client.flag = CLIENT_MULTI_STATEMENTS)
  dbWriteTable(con, "mtcars", datasets::mtcars, overwrite = TRUE)

  sql <- "SELECT cyl FROM mtcars LIMIT 5; SELECT vs FROM mtcars LIMIT 5"
  rs1 <- dbSendQuery(con, sql)
  dbFetch(rs1, n = -1)

  if (dbMoreResults(con)) {
    rs2 <- dbNextResult(con)
    dbFetch(rs2, n = -1)
  }
}
```

```

dbClearResult(rs1)
dbClearResult(rs2)
dbRemoveTable(con, "mtcars")
dbDisconnect(con)
}

```

---

dbReadTable,MySQLConnection,character-method

*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.

### Usage

```
## S4 method for signature 'MySQLConnection,character'
dbReadTable(conn, name, row.names,
  check.names = TRUE, ...)
```

```
## S4 method for signature 'MySQLConnection'
dbListTables(conn, ...)
```

```
## S4 method for signature 'MySQLConnection,character'
dbExistsTable(conn, name, ...)
```

```
## S4 method for signature 'MySQLConnection,character'
dbRemoveTable(conn, name, ...)
```

```
## S4 method for signature 'MySQLConnection,character'
dbListFields(conn, name, ...)
```

### Arguments

<code>conn</code>	a <a href="#">MySQLConnection</a> object, produced by <a href="#">dbConnect</a>
<code>name</code>	a character string specifying a table name.
<code>row.names</code>	A string or an index specifying the column in the DBMS table to use as <code>row.names</code> in the output <code>data.frame</code> . Defaults to using the <code>row_names</code> column if present. Set to <code>NULL</code> to never use row names.
<code>check.names</code>	If <code>TRUE</code> , the default, column names will be converted to valid R identifiers.
<code>...</code>	Unused, needed for compatibility with generic.

### Value

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

**Note**

Note that the data.frame returned by dbReadTable only has primitive data, e.g., it does not coerce character data to factors.

**Examples**

```
if (mysqlHasDefault()) {
  con <- dbConnect(RMySQL::MySQL(), dbname = "test")

  # By default, row names are written in a column to row_names, and
  # automatically read back into the row.names()
  dbWriteTable(con, "mtcars", mtcars[1:5, ], overwrite = TRUE)
  dbReadTable(con, "mtcars")
  dbReadTable(con, "mtcars", row.names = NULL)
}
```

---

dbUnloadDriver,MySQLDriver-method  
*Unload MySQL driver.*

---

**Description**

Unload MySQL driver.

**Usage**

```
## S4 method for signature 'MySQLDriver'
dbUnloadDriver(drv, ...)
```

**Arguments**

drv	Object created by <a href="#">MySQL</a> .
...	Ignored. Needed for compatibility with generic.

**Value**

A logical indicating whether the operation succeeded or not.

---

 dbWriteTable,MySQLConnection,character,data.frame-method

*Write a local data frame or file to the database.*


---

## Description

Write a local data frame or file to the database.

## Usage

```
## S4 method for signature 'MySQLConnection,character,data.frame'
dbWriteTable(conn, name, value,
  field.types = NULL, row.names = TRUE, overwrite = FALSE,
  append = FALSE, ..., allow.keywords = FALSE)

## S4 method for signature 'MySQLConnection,character,character'
dbWriteTable(conn, name, value,
  field.types = NULL, overwrite = FALSE, append = FALSE, header = TRUE,
  row.names = FALSE, nrows = 50, sep = ",", eol = "\n", skip = 0,
  quote = "\"", ...)
```

## Arguments

conn	a <a href="#">MySQLConnection</a> object, produced by <a href="#">dbConnect</a>
name	a character string specifying a table name.
value	a data.frame (or coercible to data.frame) object or a file name (character). In the first case, the data.frame is written to a temporary file and then imported to SQLite; when value is a character, it is interpreted as a file name and its contents imported to SQLite.
field.types	character vector of named SQL field types where the names are the names of new table's columns. If missing, types inferred with <a href="#">dbDataType</a> ).
row.names	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 identifier "row.names" maps to the DBMS (see <a href="#">make.db.names</a> ). If NA will add rows names if they are characters, otherwise will ignore.
overwrite	a logical specifying whether to overwrite an existing table or not. Its default is FALSE. (See the BUGS section below)
append	a logical specifying whether to append to an existing table in the DBMS. Its default is FALSE.
...	Unused, needs for compatibility with generic.
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 <code>mysqlWriteTable</code> to modify column names to make them legal MySQL identifiers.

header	logical, does the input file have a header line? Default is the same heuristic used by <code>read.table</code> , i.e., TRUE if the first line has one fewer column than the second line.
nrows	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.
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 <code>\</code> .)

---

isIdCurrent	<i>Check if a database object is valid.</i>
-------------	---------------------------------------------

---

### Description

Support function that verifies that an object holding a reference to a foreign object is still valid for communicating with the RDBMS. `isIdCurrent` will be deprecated in the near future; please use the `dbIsValid()` generic instead.

### Usage

```
isIdCurrent(obj)

## S4 method for signature 'MySQLDriver'
dbIsValid(dbObj)

## S4 method for signature 'MySQLConnection'
dbIsValid(dbObj)

## S4 method for signature 'MySQLResult'
dbIsValid(dbObj)
```

### Arguments

`dbObj, obj` A `MySQLDriver`, `MySQLConnection`, `MySQLResult`.

### 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.

**Examples**

```
dbIsValid(MySQL())
```

---

```
make.db.names,MySQLConnection,character-method
```

*Make R/S-Plus identifiers into legal SQL identifiers*

---

**Description**

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

**Usage**

```
## S4 method for signature 'MySQLConnection,character'
make.db.names(dbObj, snames,
  keywords = .SQL92Keywords, unique = TRUE, allow.keywords = TRUE, ...)

## S4 method for signature 'MySQLConnection'
SQLKeywords(dbObj, ...)

## S4 method for signature 'MySQLConnection,character'
isSQLKeyword(dbObj, name,
  keywords = .MySQLKeywords, case = c("lower", "upper", "any")[3], ...)
```

**Arguments**

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.
keywords	a character vector with SQL keywords, by default it is .MySQLKeywords define in RMySQL. This may be overridden by users.
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
...	Unused, needed for compatibility with generic.
name	a character vector of SQL identifiers we want to check against keywords from the DBMS.
case	a character string specifying whether to make the comparison as lower case, upper case, or any of the two. it defaults to any.



---

`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 representing the loaded client library version.

**Examples**

```
mysqlClientLibraryVersions()
```

---

`MySQLDriver-class`*Class MySQLDriver with constructor MySQL.*

---

**Description**

An MySQL driver implementing the R database (DBI) API. This class should always be initialized with the `MySQL()` function. It returns a singleton that allows you to connect to MySQL.

**Usage**

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

**Arguments**

`max.con` maximum number of connections that can be open at one time. There's no intrinsic 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.)

**Examples**

```

if (mysqlHasDefault()) {
  # connect to a database and load some data
  con <- dbConnect(RMySQL::MySQL(), dbname = "test")
  dbWriteTable(con, "USArrests", datasets::USArrests, overwrite = TRUE)

  # query
  rs <- dbSendQuery(con, "SELECT * FROM USArrests")
  d1 <- dbFetch(rs, n = 10)      # extract data in chunks of 10 rows
  dbHasCompleted(rs)
  d2 <- dbFetch(rs, n = -1)     # extract all remaining data
  dbHasCompleted(rs)
  dbClearResult(rs)
  dbListTables(con)

  # clean up
  dbRemoveTable(con, "USArrests")
  dbDisconnect(con)
}

```

---

mysqlHasDefault	<i>Check if default database is available.</i>
-----------------	------------------------------------------------

---

**Description**

RMySQL examples and tests connect to a database defined by the `rs-dbi` group in `~/my.cnf`. This function checks if that database is available, and if not, displays an informative message.

**Usage**

```
mysqlHasDefault()
```

**Examples**

```

if (mysqlHasDefault()) {
  db <- dbConnect(RMySQL::MySQL(), dbname = "test")
  dbListTables(db)
  dbDisconnect(db)
}

```

---

 result-meta

*Database interface meta-data.*


---

## Description

See documentation of generics for more details.

## Usage

```
## S4 method for signature 'MySQLResult'
dbColumnInfo(res, ...)
```

```
## S4 method for signature 'MySQLResult'
dbGetRowsAffected(res, ...)
```

```
## S4 method for signature 'MySQLResult'
dbGetRowCount(res, ...)
```

```
## S4 method for signature 'MySQLResult'
dbHasCompleted(res, ...)
```

```
## S4 method for signature 'MySQLResult'
dbGetException(conn, ...)
```

```
## S4 method for signature 'MySQLResult'
summary(object, verbose = FALSE, ...)
```

```
## S4 method for signature 'MySQLResult'
show(object)
```

## Arguments

res, conn, object	An object of class <a href="#">MySQLResult</a>
...	Ignored. Needed for compatibility with generic
verbose	If TRUE, print extra information.

## Examples

```
if (mysqlHasDefault()) {
  con <- dbConnect(RMySQL::MySQL(), dbname = "test")
  dbWriteTable(con, "t1", datasets::USArrests, overwrite = TRUE)

  rs <- dbSendQuery(con, "SELECT * FROM t1 WHERE UrbanPop >= 80")
  dbGetStatement(rs)
  dbHasCompleted(rs)
```

```

dbGetInfo(rs)
dbColumnInfo(rs)

dbClearResult(rs)
dbRemoveTable(con, "t1")
dbDisconnect(con)
}

```

---

transactions

*DBMS Transaction Management*

---

### Description

Commits or roll backs the current transaction in an MySQL connection. Note that in MySQL DDL statements (e.g. CREATE TABLE) can not be rolled back.

### Usage

```

## S4 method for signature 'MySQLConnection'
dbCommit(conn, ...)

## S4 method for signature 'MySQLConnection'
dbBegin(conn, ...)

## S4 method for signature 'MySQLConnection'
dbRollback(conn, ...)

```

### Arguments

conn	a MySQLConnection object, as produced by <a href="#">dbConnect</a> .
...	Unused.

### Examples

```

if (mysqlHasDefault()) {
con <- dbConnect(RMySQL::MySQL(), dbname = "test")
df <- data.frame(id = 1:5)

dbWriteTable(con, "df", df)
dbBegin(con)
dbGetQuery(con, "UPDATE df SET id = id * 10")
dbGetQuery(con, "SELECT id FROM df")
dbRollback(con)

dbGetQuery(con, "SELECT id FROM df")

dbRemoveTable(con, "df")
dbDisconnect(con)
}

```

# Index

- .MySQLPkgName (constants), [2](#)
- .MySQLPkgRCS (constants), [2](#)
- .MySQLPkgVersion (constants), [2](#)
- .MySQLSQLKeywords (constants), [2](#)
  
- CLIENT\_COMPRESS (constants), [2](#)
- CLIENT\_CONNECT\_WITH\_DB (constants), [2](#)
- CLIENT\_FOUND\_ROWS (constants), [2](#)
- CLIENT\_IGNORE\_SIGPIPE (constants), [2](#)
- CLIENT\_IGNORE\_SPACE (constants), [2](#)
- CLIENT\_INTERACTIVE (constants), [2](#)
- CLIENT\_LOCAL\_FILES (constants), [2](#)
- CLIENT\_LONG\_FLAG (constants), [2](#)
- CLIENT\_LONG\_PASSWORD (constants), [2](#)
- CLIENT\_MULTI\_RESULTS (constants), [2](#)
- CLIENT\_MULTI\_STATEMENTS (constants), [2](#)
- CLIENT\_NO\_SCHEMA (constants), [2](#)
- CLIENT\_ODBC (constants), [2](#)
- CLIENT\_PROTOCOL\_41 (constants), [2](#)
- CLIENT\_RESERVED (constants), [2](#)
- CLIENT\_SECURE\_CONNECTION (constants), [2](#)
- CLIENT\_SSL (constants), [2](#)
- CLIENT\_TRANSACTIONS (constants), [2](#)
- constants, [2](#)
  
- db-meta, [2](#)
- dbApply, [3](#)
- dbApply, MySQLResult-method (dbApply), [3](#)
- dbBegin, MySQLConnection-method (transactions), [20](#)
- dbClearResult, MySQLResult-method (dbFetch, MySQLResult, numeric-method), [8](#)
- dbColumnInfo, MySQLResult-method (result-meta), [19](#)
- dbCommit, MySQLConnection-method (transactions), [20](#)
- dbConnect, [8](#), [11](#), [12](#), [14](#), [20](#)
- dbConnect, MySQLConnection-method (dbConnect, MySQLDriver-method), [5](#)
- dbConnect, MySQLDriver-method, [5](#)
- dbDataType, [14](#)
- dbDataType, MySQLConnection-method (dbDataType, MySQLDriver-method), [7](#)
- dbDataType, MySQLDriver-method, [7](#)
- dbDisconnect, MySQLConnection-method (dbConnect, MySQLDriver-method), [5](#)
- dbEscapeStrings, [7](#)
- dbEscapeStrings, MySQLConnection, character-method (dbEscapeStrings), [7](#)
- dbEscapeStrings, MySQLResult, character-method (dbEscapeStrings), [7](#)
- dbExistsTable, MySQLConnection, character-method (dbReadTable, MySQLConnection, character-method), [12](#)
- dbFetch, MySQLResult, missing-method (dbFetch, MySQLResult, numeric-method), [8](#)
- dbFetch, MySQLResult, numeric-method, [8](#)
- dbGetException, MySQLConnection-method (db-meta), [2](#)
- dbGetException, MySQLResult-method (result-meta), [19](#)
- dbGetInfo, MySQLConnection-method (db-meta), [2](#)
- dbGetInfo, MySQLDriver-method, [10](#)
- dbGetInfo, MySQLResult-method (dbFetch, MySQLResult, numeric-method), [8](#)
- dbGetRowCount, MySQLResult-method (result-meta), [19](#)
- dbGetRowsAffected, MySQLResult-method (result-meta), [19](#)
- dbGetStatement, MySQLResult-method (dbFetch, MySQLResult, numeric-method), [8](#)

- dbHasCompleted, MySQLResult-method  
(result-meta), 19
- dbIsValid, 15
- dbIsValid, MySQLConnection-method  
(isIdCurrent), 15
- dbIsValid, MySQLDriver-method  
(isIdCurrent), 15
- dbIsValid, MySQLResult-method  
(isIdCurrent), 15
- dbListConnections, MySQLDriver-method  
(dbGetInfo, MySQLDriver-method),  
10
- dbListFields, MySQLConnection, character-method  
(dbReadTable, MySQLConnection, character-method),  
12
- dbListFields, MySQLResult, missing-method  
(dbFetch, MySQLResult, numeric-method),  
8
- dbListResults, MySQLConnection-method  
(db-meta), 2
- dbListTables, MySQLConnection-method  
(dbReadTable, MySQLConnection, character-method),  
12
- dbMoreResults (dbNextResult), 11
- dbMoreResults, MySQLConnection-method  
(dbNextResult), 11
- dbNextResult, 11
- dbNextResult, MySQLConnection-method  
(dbNextResult), 11
- dbReadTable, MySQLConnection, character-method,  
12
- dbRemoveTable, MySQLConnection, character-method  
(dbReadTable, MySQLConnection, character-method),  
12
- dbRollback, MySQLConnection-method  
(transactions), 20
- dbSendQuery, 4
- dbSendQuery, MySQLConnection, character-method  
(dbFetch, MySQLResult, numeric-method),  
8
- dbUnloadDriver, MySQLDriver-method, 13
- dbWriteTable, MySQLConnection, character, character-method  
(dbWriteTable, MySQLConnection, character, data.frame-method),  
14
- dbWriteTable, MySQLConnection, character, data.frame-method,  
14
- fetch, 17
- fetch, MySQLResult, missing-method  
(dbFetch, MySQLResult, numeric-method),  
8
- fetch, MySQLResult, numeric-method  
(dbFetch, MySQLResult, numeric-method),  
8
- isIdCurrent, 15
- isSQLKeyword, MySQLConnection, character-method  
(make.db.names, MySQLConnection, character-method),  
16
- make.db.names, 14
- make\_db\_names, MySQLConnection, character-method,  
16
- MySQL, 6, 9, 10, 13
- MySQL (MySQLDriver-class), 17
- mysqlClientLibraryVersions, 17
- MySQLConnection, 9, 12, 14
- MySQLDriver-class, 17
- mysqlHasDefault, 18
- MySQLResult, 9, 19
- result-meta, 19
- RMySQL (MySQLDriver-class), 17
- RMySQL-package (MySQLDriver-class), 17
- show, MySQLConnection-method (db-meta), 2
- show, MySQLDriver-method  
(dbGetInfo, MySQLDriver-method),  
10
- show, MySQLResult-method (result-meta),  
19
- SQLKeyword, MySQLConnection-method  
(make.db.names, MySQLConnection, character-method),  
16
- summary, MySQLConnection-method  
(db-meta), 2
- summary, MySQLDriver-method  
(dbGetInfo, MySQLDriver-method),  
10
- summary, MySQLResult-method  
(result-meta), 19
- transactions, 20