

# An R/S Interface to the MySQL Database

David A. James

dj@research.bell-labs.com

Bell Labs, Lucent Technologies

Murray Hill, NJ

March 1, 2001

## Contents

<b>1 Introduction</b>	<b>2</b>
<b>2 Interface Summary</b>	<b>2</b>
<b>3 Initialization</b>	<b>3</b>
<b>4 Connecting to MySQL</b>	<b>4</b>
<b>5 User Authentication</b>	<b>5</b>
<b>6 Executing SQL Statements</b>	<b>6</b>
<b>7 Convenience Import/Export Functions</b>	<b>7</b>
<b>8 Meta-data</b>	<b>8</b>
<b>9 Database Transactions</b>	<b>9</b>
<b>10 Limitations</b>	<b>10</b>
<b>11 More Examples</b>	<b>10</b>
<b>12 Resources</b>	<b>11</b>
<b>13 Acknowledgements</b>	<b>11</b>
<b>A Obtaining and Installing the Software</b>	<b>12</b>

## Abstract

We describe the interface between R and S (Plus) to the Open Source [MySQL](#) database system.

## 1 Introduction

[MySQL](#) is a mid-size, multi-platform RDBMS popular in the open source community. Some of its advantages include high-performance, open source, and free for non-commercial use. For a detailed discussion and tutorial on using relational databases with R (and hence S) see [4]; for more details on importing/exporting data into R see [5], for S-Plus see its *User's Guide*. For details on obtaining and installing the software described here see the Appendix.

The R/S MySQL interface allows multiple connections to one or more MySQL servers simultaneously (up to 100 connections). Although each MySQL connection can only have one open result set (open query) at a time, this is not a serious limitation since we can very easily clone connections (see Section 4 for details).

All functions described here are documented on-line – simply use `help(MySQL)`, for instance, to read all the details.

## 2 Interface Summary

Here we briefly illustrate how to use the interface to MySQL from either R or S. If you need or are interested in understanding the details, see the remainder of this document.

### 1. Access the library

```
> # in R 1.1.0 and later versions
> library(RMySQL)

> # in Splus5.x and later versions
> library(SMySQL)
```

(from now on, we will use `library(MySQL)` to refer generically to either the R or S implementation).

### 2. Initialize the MySQL interface:

```
> mgr <- dbManager("MySQL")
> mgr
MySQLManager id = (1455)
```

### 3. Open a connection to a database instance (see Section 5 for details on user authentication):

```
> con <- dbConnect(mgr, dbname = "opto")
> con
MySQLConnection id = (1455,1)
```

4. Meta-data – e.g., list of available tables on the database:

```
> getTables(con)
  Tables in opto
1          PBCT
2          PURGE
3           WL
4         liv25
5         liv85
```

5. Run an SQL query:

```
> rs <- dbExecStatement(con, "select * from liv25")
> rs
MySQLResultSet id (1455,1,1)
```

6. Extract all data into a data.frame:

```
> data <- fetch(rs, n = -1)
> class(data)
[1] "data.frame"
> dim(data)
[1] 1779 253
> quantile(data[, "DLDI50"])
 0% 25% 50% 75% 100%
 0 0.45 0.47 0.49 0.58
```

7. Close the result set, the connection, and unload the interface

```
> close(rs)
> close(con)
> unload(mgr)
```

Note that we created one connection, but we could have simultaneously opened connections to multiple servers and databases. Also, we provide convenience functions for importing tables into R/S and exporting data.frame objects to MySQL, see Section 7 for details.

### 3 Initialization

Following the paradigm described in the [RS-DBI \[2\]](#), we define the class `MySQLManager` that extends `dbManager` and implements the methods `load`, `unload`, `getVersion`, in addition to `show`, `describe`, and other meta-data methods.

```
> library(MySQL)
> mgr <- dbManager("MySQL")
> describe(mgr, verbose = T)
MySQLManager id = (1249)
Max connections: 16
Conn. processed: 0
```

```
Default records per fetch: 500
MySQL client version: 3.22.27
RS-DBI version: 0.2
Open connections: 0
```

Note that we create a MySQL database manager object `mgr` with a call to `dbManager`, which automatically invokes the `load` method to initialize the client part of the software (R and S play the role of clients to the database server). As we can see from the `describe` method, the `mgr` object can handle up to 16 simultaneous connections (to possibly different MySQL servers); by default the S/R client will bring over up to 500 records per fetch (see Section 6); it's based on MySQL client software version 3.22.27; and it implements the interface RS-DBI version 0.2.

Note that the `MySQLManager` object is a singleton, that is, on subsequent invocations it returns the same initialized object. Use `unload(mgr)` to free up resources once you're finished working with the MySQL database.

## 4 Connecting to MySQL

Once we've initialized the `dbManager` object by invoking the MySQL back-end we can open one or more connections to either local or remote databases; note how we use the generic function `dbConnect()` with the MySQL database manager object `mgr`:

```
> mgr <- dbManager("MySQL")
> con <- dbConnect(mgr, dbname = "opto")

# Let's look at the status of the connection
> describe(con, verbose = T)
MySQLConnection id = (1455,1)
  User: opto
  Host: localhost
  Dbdname: opto
  Connection type: Localhost via UNIX socket
  MySQL server version: 3.22.27
  MySQL client version: 3.22.27
  MySQL protocol version: 10
  MySQL server thread id: 3
  No resultSet available
```

The `dbConnect` is used to establish a connection to one database instance. Notice that you can specify the manager as a character string, "MySQL", a call to `MySQL()`, or used an existing connection object:

```
> con1 <- dbConnect("MySQL", group = "vitalAnalysis")
> con2 <- dbConnect(MySQL(), group = "vitalAnalysis")
> con3 <- dbConnect(con1)
```

the last statement simply *"clones"* `con1` – that is, it creates a new connection to the same database instance, using the same authorization used for `con1`, but it does not inherit `con1`'s result set.

In the case of MySQL, and most other RDBMS, users need to provide, at least, a login and password. Oftentimes we may also want to specify what database instance we want to connect (e.g., `opto`, `iptraffic`). You may specify all these argument in the call to `dbConnect`, but this presents some serious security risks, because both R and S are interpreted languages whose scripts need to be readable by all potential users. Therefore a somewhat more secure mechanism is needed.

## 5 User Authentication

The preferred method to pass authentication parameters to the server (e.g., user, password, host) is through the MySQL configuration file `$HOME/.my.cnf` under Unix<sup>1</sup>. Since specifying passwords on calls to `dbConnect` is a very bad idea (and so is specifying passwords through Shell variables), the client code parses the configuration file `$HOME/.my.cnf`, a file that 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 = wyner
database = iptraffic
```

This file must *not* be readable by anybody but you. Inside each section, MySQL parameters may be specified one per line (e.g., `user = opto`). The R/S MySQL implementation always parses the `[client]` and `[rs-dbi]` sections, but you may define you 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").
```

---

<sup>1</sup> For Windows, the MySQL documentation describes the file `c:\my.cnf`

The most important authentication parameters in `$HOME/.my.cnf` are `user`, `password`, `host`, and `database`. Note that the database name cannot go in the `[client]` section, but you may safely include it under the `[rs-dbi]` section or one you define yourself. (See [1].)

```
# open the connection using user, password, etc., as
# specified in the "[iptraffic]" section of the
# configuration file file $HOME/.my.cnf
> con <- dbConnect(mgr, group = "iptraffic")

# open connections to the distributed opto database
con1 <- dbConnect(mgr, dbname = "opto", host = "wyner")
con2 <- dbConnect(mgr, dbname = "opto", host = "merced")
```

Make sure you close the connection using `close(con)` when it is no longer needed. Notice that you will not be able to close a connection that has pending rows to fetch, see next section.

## 6 Executing SQL Statements

Once we have an open connection to a database in a MySQL server we can execute SQL statements. The methods `dbExecStatement`, `dbExec`, and `quickSQL` send their input to the server for execution. In case there is any error (SQL syntax, for instance) the method fails and prints an error message. You can extract the status of the last SQL statement by invoking `getException` on the appropriate connection object.

If there is no error, the above methods return an object of class `dbResult`, describing the status of the operation (for instance, the number of rows affected). If the SQL operation produced an output set (for instance a `SELECT` statement) the output will extend `dbResultSet` in addition to `dbResult`, and you may use the `fetch` method to bring over the output rows into `S` or `R`.

```
# Run an SQL statement by first creating a resultSet object
> rs <- dbExecStatement(con,
  statement = "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")

> rs
MySQLResultSet id = (1455,1,3)

# we now fetch the first 1000 records
> data1 <- fetch(rs, n = 1000)
> dim(data)
[1] 1000 18

> hasCompleted(rs)
```

```
[1] 0

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

You can also close a result set to discard pending rows. Notice, however, that you will not be allowed to close a *connection* that has a result set with pending rows – you need to either fetch all the rows or close the result set before closing the connection.

The `quickSQL` is a support function that combines the operation of executing a statement and fetching its output, thus it makes most sense for SELECT-like statements.

## 7 Convenience Import/Export Functions

The function `getTable()`, `removeTable()`, `existsTable()`, and `assignTable()` mimic their R/S counterpart `get()`, `remove()`, `exists()`, and `assign()` (recall that `getTables()` lists all the remote tables available on a connection). These functions all take a database connection object and an SQL table name (plus a `data.frame` in the case of `assignTable()`) to import into R/S, delete, check that the table exists, and to export a `data.frame` to the DBMS.

```
> # R code
> data(USArrests)
> con <- dbConnect("MySQL", user = "test", dbname = "test")
> existsTable(con, "US_Arrests")
[1] FALSE
> assignTable(con, "US_Arrests", USArrests)

> con2 <- dbConnect("MySQL", group = "iptraffic")
> trantime <- getTable(con, "TRANTIME")
```

These functions allows us to *"attach"* databases to our `S` search path and access the database transparently (this functionality is not yet available in R):

```
S> vcon <- dbConnect("MySQL", group = "vitalAnalysis")
S> a <- attach(vcon, max.rows=25000, translate = T)
S> ls(pos=2)
(pos=2)[1:5]
[1] "AGGREGATE.APP.TRANTABLE"
[2] "AGGREGATE.APPSRVRPROXY.HTTPTRANTABLE"
[3] "AGGREGATE.GRPAPPSRVRPROXY.HTTPTRANTABLE"
[4] "CLIENTMAPTABLE"
[5] "DNSTRANTABLE"

> names(CLIENTMAPTABLE)
[1] "CLIENTID"      "CLIENTNAME"    "LASTIPADDRESS"
[4] "LASTENTRYROUTER" "TIMEZONE"      "OS"
.....
[16] "AGENTLICENSES" "AGENTVERSION"  "LASTGROUPID"
```

The argument `max.rows=25000` prevents tables with more than this many rows to be imported; the argument `translate=T` causes all SQL identifiers to be translated to valid S identifiers. (For more details see[3].)

## 8 Meta-data

One very useful property of RDBMS is that their data are self-describing (as you already know, data in S and R are also self-describing). That is, one can dynamically query information about a database, its tables, field names and field types, what indices are defined on these tables, etc. The R/S interface to databases defines methods for querying these meta-data, and you'll notice that these methods mimic the R/S notions of `objects`, `names`, `class`, `mode`, etc. These meta-data methods allow us to navigate the database to locate tables, fields, etc.

More generally, we recognize two types of meta-data: one type that describes the interface between R/S and MySQL, and another type that describes the database objects themselves. The methods `describe` and `version` extract meta-data relevant to the interface software, while all other methods are relevant to database objects.

```
> describe(mgr, verbose = F)
MySQLManager id = (1455)
  Max connections: 16
  Conn. processed: 1
  Default records per fetch: 500
  Open connections: 1
> getVersion(mgr)
$"RS-DBI":
[1] "0.2"

$"MySQL (client) library":
[1] "3.22.27"
> getFields(rs)
      name      Sclass      type len precision scale nullok
1 Tables in opto character FIELD.TYPE.STRING 64      64      0      F

> describe(rs)
MySQLResultSet id = (1455,1,4)
  Statement: show tables
  Has completed? no
  Affected rows: -1
  Rows fetched: 1779
```

Meta-data that the MySQL server makes available includes lists of databases, lists of tables, field description for each table, lists of indices defined on each table, etc. Be aware that some of this information may not be available to all users due to security constraints defined by the database administrator. But this issue is not too different from the Unix security model in which users have limited privileges dictated by user login and group membership. The most important meta-data include

`getVersion(obj)` produces a list with the version `obj` implements;

`getDatabases(mgr, ...)` list of available databases;

`getCurrentDatabase(con)` the name of the database current in the connection object `con`;

`getTables(mgr, dbname, ...)` list of tables in database `dbname`;

`getTableIndices(con, table, dbname, ...)` list of indices for `table` in database `dbname`

`getException(con)` list with exception number and description of the last operation performed on connection `con`;

`getResultSet(con)` returns the result set associated with connection `con`;

`getStatement(rs)` returns the SQL statement associated in the result set `rs`;

`getDBconnection(rs)` returns the connection object associated with the result set `rs`;

`getFields(rs)` returns a `data.frame` that fully describes the fields of the extracted rows and available through `rs`; the `data.frame` has 7 columns (field name, R/S data type, MySQL data type, length, precision, scale, and null flag) and each row describes one field in the result set;

`getFields(mgr, table, dbname, ...)` returns a description of the fields in `table` in database `dbname`;

`hasCompleted(rs)` a logical describing whether the SQL instruction has completed or not (SELECT statements are considered incomplete so long as there are records to fetch);

`getRowCount(rs)` number of rows fetched so far from result set `rs`

`getRowsAffected(rs)` number of rows affected by last SQL statement (most useful for non-SELECT statements like DELETE, INSERT);

`getNullOk(rs)` a logical vector describing which fields in the result set `rs` accept NULL values;

`getInfo(dbObject, what)` extract the field `what` from the meta-data list associated with the object `dbObject`. A `dbObject` refers to any R/S database object reference, i.e., `dbManger`, `dbConnections`, `dbResult` or `dbResultSet`.

## 9 Database Transactions

Since the MySQL system does not define transactions (as of version 3.23) the R/S implementation simply ignores calls to `commit` and `rollback`.

## 10 Limitations

Note that the current MySQL interface has not implemented the `setDataMapping` method for `dbResultSet` objects, and in particular, date and other time fields are returned as character strings — users need to convert them to their favorite data/time classes of objects. Also, the implementation inappropriately overloads the functionality of the `dbResult` class in the class `resultSet`.

## 11 More Examples

Some more examples:

```
# Extract meta-data information.  What MySQL databases
# are there available on host "wyner"
> getDatabases(m, host = "wyner")
  Database
1      mysql
2      opto
3      test
4 iptraffic
5      fraud

# What tables are there in the "opto" database?

> dbTables(m, dbname = "opto", host = "wyner")
  Tables in opto
1          PBCT
2          PURGE
3           WL
4        liv25
5        liv85

# let's look at some result set meta-data

> con <- dbConnect(mgr)
> rs <- dbExecStatement(con, query.sql)

> getStatement(rs)
[1] "show tables"

> hasCompleted(rs)
[1] 0

> getRowCount(rs)
[1] 3

> info <- getInfo(rs)
> names(info)
[1] "statement"      "isSelect"      "rowsAffected"
```

```

[4] "rowCount"      "completed"      "fieldDescription"

# the following are pieces of meta-data associated with
# the R/S DBI implementation, versions for the various pieces
# of software (client, server, interface), etc.

# dbManager object

> names(getInfo(m))
[1] "connectionIds"      "fetch_default_rec"
[3] "managerId"          "length"
[5] "num_con"            "counter"
[7] "clientVersion"

# dbConnection object

> names(getInfo(con))
[1] "host"              "user"
[3] "dbname"            "conType"
[5] "serverVersion"     "protocolVersion"
[7] "threadId"          "rsId"

# resultSet object

> names(getInfo(rs))
[1] "statement"          "isSelect"
[3] "rowsAffected"       "rowCount"
[5] "completed"          "fieldDescription"

```

See the on-line documentation for more details.

## 12 Resources

This implementation follows the proposed R/S Database Interface (DBI); the latest documentation and software is available at [www.omegahat.org](http://www.omegahat.org). The R community has developed interfaces to some databases: [RmSQL](#) is an interface to the [mSQL](#) database written by Torsten Hothorn; [RPgSQL](#) is an interface to [PostgreSQL](#) and was written by Timothy H. Keitt; finally, [RODBC](#) is an interface running under Windows to ODBC, and it was written by [Michael Lapsley](#).

## 13 Acknowledgements

[Doug Bates](#) and [Saikat DebRoy](#) ported (and greatly improved) the first MySQL implementation to R.

The R/S database interface came about from suggestions, comments, and discussions with [John M. Chambers](#) and [Duncan Temple Lang](#) in the context of the Omega Project for Statistical Computing.

## A Obtaining and Installing the Software

You may obtain the software either from <http://www.omegahat.org/contrib/RS-DBI>, <http://stat.bell-labs.com/stat/RS-DBI>, or <http://cran.r-project.org>.

To use RMySQL you will need R version 1.2.0 or later; to install, simply type

```
R CMD INSTALL RMySQL_<version>.tar.gz
```

where RMySQL\_<version>.tar.gz is the file you obtained from one of the above places.

To use SMySQL you will need Splus version 5.0 or later; to install, untar the file SMySQL\_<version>.tar.gz and follow the instructions in the file `Installation`.

The version (for R and S) as of March 1st, 2001, is 0.4.1.

## References

- [1] Paul DuBois. *MySQL*. New Riders Publishing, 2000. 1
- [2] David A. James. A common interface to relational database management systems from r and s — a proposal. Technical report, Bell Labs, Lucent Technologies, [www.omegahat.org](http://www.omegahat.org), 2000. 3
- [3] David A. James. Proxy database object in the s language. Technical report, Bell Labs, Lucent Technologies, In preparation. 7
- [4] R-Development Core Team. *Using Relational Database Systems with R*, 2000. Draft. 1
- [5] R-Development Core Team. *R Data Import/Export*, 2001. Version 1.2.1 (2001-01-15). 1