Using Relational Database Systems with R

Version 0.99.0

R Development Core Team

Table of Contents

Acknowledgments

The contributions of David James and John Chambers, who drafted the database applications programming interface for S and R, are gratefully acknowledged. Implementations of the API were created by David James and Saikat DebRoy (MySQL) and by Timothy Keitt (PostgreSQL).

1 Introduction

The R system provides a versatile environment for statistical analysis and graphical presentation of data. More generally, it provides facilities for "programming with data", to use John Chambers' term.

However, there are limitations on the types of data that R handles well. Since all data being manipulated by R are resident in memory, and many copies of the data can be created during execution of a function, R is not well suited to extremely large data sets. Data objects that are more than a few megabytes in size can cause R to run out of memory.

R does not easily support concurrent access to data. That is, if more than one user is accessing, and perhaps updating, the same data, the changes made by one user will not be visible to the others.

R does support persistence of data, in that you can save a data object or an entire worksheet from one session and restore it at the subsequent session, but the format of the stored data is specific to R and not easily manipulated by other systems.

Database management systems (DBMS's) and, in particular, relational database management systems such as Oracle, Sybase, MySQL, and PostgreSQL are designed to do all of these things well. They are designed to provide fast access to selected parts of large databases. The can provide concurrent access from multiple clients running on multiple hosts while enforcing security constraints on access to the data. Furthermore the same data can be accessed by many different kinds of clients.

Rather than recreate all the facilities of database management systems, we provide interfaces to relational database systems from R.

2 Relational database management systems (RDBMS)

A relational database management system stores data in tables or 'relations'. The corresponding structure in R is a data frame. All the values in a column must have exactly the same representation, whether this is as a string of characters or as a specific type of numerical value, such as a 16-bit unsigned integer. Rows can be inserted, deleted or modified. Information from several tables can be combined to create new tables.

Structured Query Language, or SQL, is a standard language used by most RDBMS to declare tables and to manipulate the data in tables. There are several RDBMS available including commercial products such as Oracle (www.oracle.com), Sybase (www.sybase.com), DB/2 (www.ibm.com). The PostgreSQL system (www.postgresql.org) is freely available. The MySQL clients (www.mysql.com) are freely available as is the MySQL server for Linux/Unix systems. A flat fee is charged for the MySQL server for Windows NT.

There are many good books on SQL and on particular relational database systems. Those of us who use MySQL have found Dubois (2000) to be very helpful.

2.1 Description of data for an example

To demonstrate creating tables and manipulating data in tables, we consider a specific example. We will use data on Mathematics Achievement scores reported in chapter 4 of Bryk and Raudenbush (1992). These data are a subset of the data from the 1982 *High School and Beyond Survey*.

The data consist of information on schools, including the ID number for the school, the size (number of students), the sector (Public or Catholic), and the proportion of students in an academic track; and information on individual students, including the school they attend, whether or not the student is a member of a racial minority, the student's sex, a measure of Socio-Economic Status (SES) and the difference in their Mathematics Achievement scores from the beginning of the year to the end of the year. There are data on 7185 students in 160 different schools.

We create two tables, one with the information on the schools and one with the information on the students within the schools.

2.2 Creating tables

In `SQL` the `CREATE TABLE` statement is used to create a table. Most RDBMS require, and can take advantage of, considerable detail in specifying the contents and storage types of the columns. One can distinguish between a `SMALLINT` (usually 16 bits) that saves storage space at the expense of having a more limited range and an `INT`. One should also indicate whether the values can be `NULL`. A `NULL` value in `SQL` corresponds to a missing value (`NA`) in R.

We can distinguish certain columns as `key`s for which indices should be created to allow faster lookups.

Unlike in R, identifiers and keywords in `SQL` are not case sensitive.

Although there is a specification for a base `SQL` language, most RDBMS allow extensions. The examples we show here were done with MySQL and may use extensions peculiar to MySQL.

The SQL to declare a table for the schools data is

```
create table school (id smallint unsigned not null primary key,
   size smallint unsigned not null,
   sector enum('Public', 'Catholic') not null,
   pracad float not null,
   disclim float not null,
   himinty tinyint unsigned not null,
   meanses double not null)
```

This states that the table will be called schools and will contain six columns. The first column, id, will be an unsigned small integer used as the primary key. The values for the primary key must be unique and must not be NULL.

MySQL allows data to be loaded in a single step with the LOAD DATA statement. We previously constructed a file called /tmp/School.dat containing the data for this table in tab-separated fields. We load it with

```
mysql> load data infile '/tmp/School.dat' into table school;
Query OK, 160 rows affected (0.01 sec)
Records: 160  Deleted: 0  Skipped: 0  Warnings: 0
```

The basic command for retrieving data in SQL is the SELECT statement. To check that the data have been properly entered we select all the columns from the table but limit the result to the first 10 rows.

```
mysql> select * from school limit 10;
+------+------+----------+--------+---------+---------+---------+
| id   | size | sector   | pracad | disclim | himinty | meanses |
+------+------+----------+--------+---------+---------+---------+
| 1224 |  842 | Public   |   0.35 |    1.60 |       0 | -0.4280 |
| 1288 | 1855 | Public   |   0.27 |    0.17 |       0 |  0.1280 |
| 1296 | 1719 | Public   |   0.32 |   -0.14 |       1 | -0.4200 |
| 1308 |  716 | Catholic |   0.96 |   -0.62 |       0 |  0.5340 |
| 1317 |  455 | Catholic |   0.95 |   -1.69 |       1 |  0.3510 |
| 1358 | 1430 | Public   |   0.25 |    1.53 |       0 | -0.0140 |
| 1374 | 2400 | Public   |   0.50 |    2.02 |       0 | -0.0070 |
| 1433 |  899 | Catholic |   0.96 |   -0.32 |       0 |  0.7180 |
| 1436 |  185 | Catholic |   1.00 |   -1.14 |       0 |  0.5690 |
| 1461 | 1672 | Public   |   0.78 |    2.10 |       0 |  0.6830 |
+------+------+----------+--------+---------+---------+---------+
10 rows in set (0.01 sec)
```

We create, populate, and check a table for the students' data with

```
mysql> create table student (sch smallint unsigned not null,
    ->     minority enum('Yes', 'No') not null,
    ->     sex enum('Female', 'Male') not null,
    ->     sestat float not null,
    ->     achieve float not null,
    ->     meanses float not null,
    ->     index (sch));
Query OK, 0 rows affected (0.00 sec)

mysql> load data infile '/tmp/Student.dat' into table student;
```

```
Query OK, 7185 rows affected (0.32 sec)
Records: 7185  Deleted: 0  Skipped: 0  Warnings: 0

mysql> select * from student limit 10;
+------+----------+--------+--------+---------+---------+
| sch  | minority | sex    | sestat | achieve | meanses |
+------+----------+--------+--------+---------+---------+
| 1224 | No       | Female |  -1.53 |    5.88 |   -0.43 |
| 1224 | No       | Female |  -0.59 |   19.71 |   -0.43 |
| 1224 | No       | Male   |  -0.53 |   20.35 |   -0.43 |
| 1224 | No       | Male   |  -0.67 |    8.78 |   -0.43 |
| 1224 | No       | Male   |  -0.16 |   17.90 |   -0.43 |
| 1224 | No       | Male   |   0.02 |    4.58 |   -0.43 |
| 1224 | No       | Female |  -0.62 |   -2.83 |   -0.43 |
| 1224 | No       | Male   |  -1.00 |    0.52 |   -0.43 |
| 1224 | No       | Female |  -0.89 |    1.53 |   -0.43 |
| 1224 | No       | Male   |  -0.46 |   21.52 |   -0.43 |
+------+----------+--------+--------+---------+---------+
10 rows in set (0.01 sec)
```

In this table we have several students in the same school (the `sch` column) so we cannot use `sch` as a primary key. We can, however, create an index for this column to allow faster cross-referencing of data in this table with the data in the `school` table.

2.3 Joining tables

One of the basic operations with relational databases is relating the information in different tables according to information that is common to both. This is called 'joining' tables.

Our `student` table contains redundant information on the mean socio-economic status for the school that the student attends. We store this information in the school table **and** for each student, even though it is a property of the school and not a property of the student. We can avoid this redundancy by selecting information for the school at the same time as we select the student information.

```
mysql> select t.sch, t.minority, t.sex, t.sestat, t.achieve, s.meanses
    ->    from student as t, school as s
    ->    where t.sch = s.id
    ->    limit 10;
+------+----------+--------+--------+---------+---------+
| sch  | minority | sex    | sestat | achieve | meanses |
+------+----------+--------+--------+---------+---------+
| 1224 | No       | Female |  -1.53 |    5.88 | -0.4280 |
| 1224 | No       | Female |  -0.59 |   19.71 | -0.4280 |
| 1224 | No       | Male   |  -0.53 |   20.35 | -0.4280 |
| 1224 | No       | Male   |  -0.67 |    8.78 | -0.4280 |
| 1224 | No       | Male   |  -0.16 |   17.90 | -0.4280 |
| 1224 | No       | Male   |   0.02 |    4.58 | -0.4280 |
| 1224 | No       | Female |  -0.62 |   -2.83 | -0.4280 |
| 1224 | No       | Male   |  -1.00 |    0.52 | -0.4280 |
```

```
| 1224 | No       | Female |  -0.89 |    1.53 | -0.4280 |
| 1224 | No       | Male   |  -0.46 |   21.52 | -0.4280 |
+------+----------+--------+--------+---------+---------+
10 rows in set (0.00 sec)
```

The value in the last column of this result is extracted from the school table, not from the student table, which is why it is recorded at -0.4280 and not -0.43. The data files from which these tables are derived had rounded those values in the students table.

Because we can join the information from different tables we only need to store student-specific information in the student table. Any information that is common to all the students in the school can be obtained from the school table.

2.4 Summarizing data in SQL

More complicated selects statements are possible (and common). We can use the RDBMS to summarize information in a table in different ways. For example, we can determine the number of females and males in the study.

```
mysql> select sex, count(*) from student group by sex;
+--------+------------+
| sex    | count(*) |
+--------+------------+
| Female |       3795 |
| Male   |       3390 |
+--------+------------+
2 rows in set (0.05 sec)
```

Even more interesting, we can average the socio-economic status values within schools and thereby check the `meanses` values in the table. Doing so shows that some of the values in the original data files we received are inaccurate.

```
mysql> select sch, avg(sestat) from student group by sch limit 10;
+------+-------------+
| sch  | avg(sestat) |
+------+-------------+
| 1224 |   -0.436383 |
| 1288 |    0.119600 |
| 1296 |   -0.427500 |
| 1308 |    0.526000 |
| 1317 |    0.343333 |
| 1358 |   -0.021667 |
| 1374 |   -0.014643 |
| 1433 |    0.710000 |
| 1436 |    0.560909 |
| 1461 |    0.675455 |
+------+-------------+
10 rows in set (0.01 sec)
```

3 Interfacing R to RDBMS

There are libraries available to allow access to data in several popular RDBMS from within R. The `RMySQL` library, by David James and Saikat DebRoy, follows the S/R API for database management systems. As such, it defines functions for connecting to a database, for executing `SQL` statements and for accessing the result set.

There are several sophisticated facilities available in these interfaces but it is best to start simply by creating the table you want with `SQL` then loading the table into R as a data frame.

An initial analysis of the mathematics achievement data may focus on the school that the student attends, the mean socio-economic status for the school, the student's sex, and the math achievement score.

We can load such a table into R and convert it to a data frame with

```
library(RMySQL)
m <- MySQL()       # a database system object
con <- dbConnect(m, password = 'secret', dbname = 'MathAch')
math <-
  data.frame(dbExec(paste('select t.sch, c.meanses, t.sex, t.achieve',
    'from student as t, school as c where t.sch = c.id')))
math$sch <- as.factor(math$sch)
```

The `MySQL` function creates a database system object for a MySQL database. The `dbConnect` function connects to a database administered by that RDBMS. Multiple simultaneous connections can be used, if required. The `dbExec` function executes an `SQL` statement and returns the result in a single step. It should only be used when you know that the result will be of a manageable size. In this case we know that there will be 7185 rows, which is large but not excessively large. The related function, `dbExecStatement`, allows finer control over the retrieval of data because it causes the `SQL` statement to be executed but does not retrieve all the values at once. You can retrieve sections of the result set.

Function and variable index

(Index is nonexistent)

Concept index

(Index is nonexistent)

Appendix A References

P. Dubois (2000), *MySQL.* New Riders.

A. Bryk and S. Raudenbush (1992), *Hierarchical Linear Models for Social and Behavioral Research.* Sage.