Using Databases with R

by Brian D. Ripley

There has been a lot of interest in the inter-working of R and database management systems (DBMSs) recently, and it is interesting to speculate why now. Corporate information has been stored in mainframe DBMSs for a few decades, and I remember colleagues in the 1970s using SAS (via punched cards) to extract and process information from tapes written from such databases. I guess what has changed is accessibility: personal DBMSs are widely available and no longer need teams of experts to manage them (although they still help). The *R Data Import/Export* manual introduced in version 1.2.0 provides a formal introduction to most of the facilities available, which this article supplements by some further motivation, as well as allowing for personal views.

We can look at the interfaces from two viewpoints:

1. You have data you want to organize and extract parts of for data analysis. You or your organization may find it convenient to organize the data as part of a database system, for example to help ensure data integrity, backup and audit, or to allow several people simultaneously to access the data, perhaps some adding to it and others extracting from it.

   The need can be as simple as to have a data entry and verification system.

2. You want to do some simple data manipulations for which R is not particularly suitable. Someone on `r-help` wanted to do a merge on 30,000 rows. DBMSs are (usually) very good at merges, but R's `merge` is only designed for small-scale problems. Rather than spend time re-writing `merge`, why not use an already-optimized tool?

Choosing a DBMS

A wide range of DBMSs are available, and you may already have one in use. If not, the most popular contenders on Unix/Linux appear to be PostgreSQL (`http://www.postgresql.org/`) and MySQL (`http://www.mysql.com/`). PostgreSQL is Open Source and competitive on features and standards-conformance with the commercial big names (many of which have 'free' lightweight Linux versions). MySQL is 'lean and mean' but with limited security features, little transaction support, ...

On Windows, the most popular database of any sophistication is undoubtedly Access, which has a big brother SQL Server. Pre-compiled development versions of MySQL can be downloaded and were used for most of our testing under Windows. It is possible[8] to build and use PostgreSQL under the Cygwin environment.

I will assume that we are working with a *relational* database. These store the data in a collection of *tables* (also known as *relations*) which are closely analogous to R's data frames: they are conceptually rectangular made up of columns (or 'fields') of a single[9] type, for example character, monetary, datetime, real, ..., and rows ('records') for each case.

The common DBMSs are client-server systems, with a 'backend' managing the database and talking to one or more clients. The clients can be simple command-line interfaces[10] such as provided by `mysql` and `psql`, general-purpose or application-specific GUI clients, or R interfaces as discussed here. The clients communicate with the backend by sending requests (usually) in a dialect of a language called SQL, and receive back status information or a representation of a table.

Almost all of these systems can operate across networks and indeed the Internet, although this is often not enabled by default.

Choosing an interface

There are several ways to interface with databases. The simplest is the analogue of 'sneaker LAN', to transfer *files* in a suitable format, although finding the right format may well not be simple.

Four contributed R packages providing interfaces to databases are described in *R Data Import/Export*, but two are currently rather basic. By far the most portable option is **RODBC**, and unless you can choose your DBMS you probably have no other option. Open Database Connectivity (ODBC) is a standard originally from the Windows world but also widely available on Linux. It provides a common client interface to almost all popular DBMSs as well as other database-like systems, for example Excel spreadsheets. To use ODBC you need a driver manager for your OS and a driver for that and your DBMS. Fortunately drivers are widely available, but not always conforming to recent versions[11] of ODBC.

The basic tools of **RODBC** are to transfer a data frame to and from a DBMS. This is simple: use commands like

```
sqlSave(channel, USArrests, rownames = "state")
```

[8] but not easy, and I know of no pre-compiled distribution.
[9] with an important exception: see 'tips' below
[10] sometimes called 'monitors'
[11] that for MySQL is rather out of date.

 ISSN 1609-3631

```
sqlFetch(channel, "USArrests", rownames = TRUE)
```

to copy the R data frame to a table[12] in the database, or to copy the table to an R data frame. However, if we want to do more than use the database as a secure repository, we need to know more. One thing we can do is ask the DBMS to compute a subset of a table (possibly depending on values in other tables), and then retrieve the result (often known as a *result set*). This is done by `sqlQuery`, for example (all on one line)

```
sqlQuery(channel,
  "select state, murder from USArrests
   where rape > 30 order by murder")
```

which is the SQL equivalent of the R selection

```
z <- USArrests[USArrests$Rape > 30,
               "Murder", drop = FALSE]
z[order(z[,1]), drop = FALSE]
```

Indeed, almost anything we want to do can be done in this way, but there are shortcuts to a lot of common operations, including `sqlFetch` and `sqlSave`. As another example, let us perform the (tiny) `merge` example in a database.

```
sqlSave(channel, authors)
sqlSave(channel, books)
sqlQuery(channel,
  "SELECT a.*, b.title, b.otherauthor
   FROM authors as a, books as b
   WHERE a.surname = b.name")
```

This is of course pointless, but the technique it illustrates is very powerful.

There is a package[13] **RPgSQL** that provides a sophisticated interface to PostgreSQL. This provides analogues of the facilities described for **RODBC**. In addition it has the powerful notion of a *proxy data frame*. This is an object of an R class which inherits from data frames, but which takes little space as it is a *reference* to a table in PostgreSQL. There will be an advantage when we are accessing smaller parts of the data frame at any one time, when the R indexing operations are translated to SQL queries so that subsetting is done in PostgreSQL rather than in R.

Traps and tips

Things are not quite as simple as the last section might suggest. One problem is case, and one solution is only ever to use lowercase table and column names. As the mixtures of cases in the examples thus far suggest, many DBMSs have problems with case. PostgreSQL maps all names to lowercase, as does MySQL on Windows but not on Linux, Oracle maps all to uppercase and Access leaves them unchanged!

In a similar way the R column `other.author` was changed to `otherauthor` in the system used for the `merge` example.

Another problem is row-ordering. It is safest to regard the rows in a table as unordered, in contrast to a data frame, and indeed the optimization process used in executing queries is free to re-order the results. This means that we do need to carry along row names, and this is done by mapping them (more or less transparently) to a column in the table. It also explains why we sorted the results in the `sqlQuery` example.

Care is needed over missing values. The one exception to the rule that all entries in a column of a table must be of a single type is that an entry is allowed to be `NULL`. This value is often used to represent missing values, but care is needed especially as the monitors do not visually distinguish an empty character field from a `NULL`, but clients should. Michael Lapsley and I re-wrote **RODBC** to handle all the possibilities we envisaged.

We have so far given no thought to efficiency, and that is how it should be until it matters. Tuning databases is an art: this involves judiciously creating indices, for example. Hopefully research advances in database systems will percolate to more intelligent internal tuning by widely available DBMSs.

Another issue is to take care over is the size of the result set. It is all too easy to write a query that will create a result set that far exceeds not just the available RAM but also the available disc space. Even sensible queries can produce result sets too large to transfer to R in one go, and there are facilities to limit queries and/or transfer result sets in groups.

The future

Hopefully one day in the not too far distant future the R interfaces will be much more similar than at present, but that does depend on contributors looking at each other's designs.

It will be good to see more use of R's various types and classes, for example for times and monetary amounts.

There is a further fascinating possibility, to embed R inside a DBMS. Duncan Temple Lang mentions embedding R in PostgreSQL in a document[14] on the R developer's Web site. I don't have access to such a system, but his description is

> The initial example [...] was embedding R within PostgreSQL for use as a procedural language. This allows (privileged) users to define SQL functions as R functions, expressions, etc.

[12] probably named in lower case as here.
[13] which as far as I know it has only been used under Linux/Unix.
[14] http://developer.r-project.org/embedded.html

from which I gather that the (already rather extended) dialect of SQL can be extended by user-defined functions that call R to compute new columns from old ones.

We are exploring R–DBMS interfaces in data mining applications. There databases can be large but are perhaps growing slower than computing power. For example, insurance databases already cover around 30 million drivers. The idea is to use the DBMS not just to extract subsets for analysis but also perform cross-tabulations and search for exceptions.

Brian D. Ripley
University of Oxford, UK
ripley@stats.ox.ac.uk

Rcgi 4: Making Web Statistics Even Easier

by M.J. Ray

Webservers allow browsers to run a selection of programs chosen by the webmaster, via the Common Gateway Interface which defines how inputs are passed from browser to program and output passed back again. Naturally, it is possible to make R one of the available programs by using a "wrapper script" to translate between standard R input and outputs and CGI. Rcgi is such a script and has just been revised for a new, more powerful and easier-to-install release.

Benefits

Rcgi has been developed in response to a perceived need at the University of East Anglia. Even though we can provide students with their own copies of the R software for free, there are two principal reasons why it is still useful to provide access to our installation over the internet.

The first is that not all students have their own workstations. While this may change eventually, many students use the campus's open access computing facilities. Our department can manage software on only a small proportion of these machines, so for the others a web interface is simpler to access and insulates us from configuration changes beyond our control. As long as the computer can still run a web browser, they can use Rcgi.

Feedback from students on our third-year course (who were the first to use the system) suggests that the web front-end is popular partly because of its "batch mode" of operation, running some commands and returning the output, together with the commands for editing and resubmission.

The second and increasingly important benefit of Rcgi is the ability for the lecturer to provide worked examples to the students with the option of leaving spaces for the students to contribute their own data to the examples. Rather than having to write their own CGI programs for each example, lecturers need only write the HTML for the page (which many know already) and the program in the R language.

Example

Take the following snippet of R code, which defines a vector of numbers and generates the basic summary statistics for them:

```
test <- c(1,45,2,26,37,35,32,7,
          4,8,42,23,32,27,29,20)
print(summary(test))
```

which has the output

```
Min. 1st Qu  Median  Mean 3rd Qu.  Max.
1.00   7.75   26.50 23.13   32.75 45.00
```

For an elementary statistics course, the lecturer may wish to provide this as a worked example, but allow the students to change the values in the "test" list. The HTML code to do this is:

```
<form  method='post'
       action='/cgi-bin/Rcgi'>
<input type='hidden' name='script'
       value='test <- c(' />
<input type='text' name='script'
value='1,45,2,26,37,35,32,7,
       4,8,42,23,32,27,29,20' />
<input type='hidden'
       name='script' value=')
       print(summary(test))
       ' />
<input type='submit' value='go!' />
</form>
```

and a view of the data entry page and the results returned from Rcgi are included in figure 4. Hopefully by the time that this article appears, this example will be back online at the Rcgi site (address below).

Note that the code executed is displayed directly below the output, offering the student the chance to examine and modify the R program. Hopefully, they can learn from this experimentation in a similar way to having the entire system on their own computer.

The most commonly suggested point for improvement was the installation procedure, which was previously completely manual. The usual Unix `make` program is used to install the new release, with a script that attempts to locate the various