drdobbs.com

RDB -- A UNIX Command-Line Database

By Mark Pruett, April 10, 2003

16-20 minutes

Sometimes, even a small database is too big. Databases such as mySQL and PostgreSQL, while lightweight and easy to install by commercial standards, may still be more than you need for some straightforward data analysis problems. Some of these databases may also require root access to install, which may not be practical in some environments.

Mark Pruett

Sometimes, even a small database is too big. Databases such as mySQL and PostgreSQL, while lightweight and easy to install by commercial standards, may still be more than you need for some straightforward data analysis problems. Some of these databases may also require root access to install, which may not be practical in some environments.

RDB, a relational database system that operates on plain text files using the UNIX command line, is a simple but powerful alternative. It can be quickly downloaded and installed and doesn't require root access. Furthermore, it doesn't use a daemon process to mediate between the data tables and the user.

Walter Hobbs of the RAND Corporation wrote RDB, based on ideas

from an article published in the March 1991 issue of *Unix Review* magazine ("A 4GL Language"), and from the book *UNIX Relational Database Management*, by Rod Manis, et. al. (Prentice Hall PTR/Sun Microsystems Press). Walter Hobbs has released RDB to the public domain, so it's freely distributable.

RDB Files

RDB differs from most other relational databases in several ways. One difference is that RDB stores data tables as ASCII text files. A more traditional database system, like PostgreSQL, stores data tables in a binary format in files that are hidden from the user. The user accesses the data indirectly, through special programs or through APIs (Application Programming Interfaces).

While insulating the user from direct access to the data files has advantages, it also makes that data more difficult to move from place to place. With RDB, data tables can be moved anywhere, even to other computers, by simply copying the data files. Other relational databases often require you to export the table (ironically, to a text file), then import the file into the new database. RDB tables can be created with any text editor. Figure 1 (people.rdb) shows an example of a simple RDB table.

The first line of the table in Figure 1 is a comment. Comment lines are optional and there can be more than one, but they must come at the beginning of the file, and they must begin with a "#" character. The second line describes the table's column names. Starting with the second line, all data columns are separated by a tab character. In Figure 1, tab characters are denoted by "<t>". The people.rdb table has four columns: LastName, FirstName, Age, and Phone.

The next line describes the table's column types. Traditional

databases have many column types, including strings, integers of varying widths, floating-point numbers, dates, and even BLOBs (Binary Large OBjects). RDB has relatively few: string, numeric, and month. The people.rdb table contains mostly string columns, with Age being the lone numeric field. Each tab-separated value describes the column type for its corresponding column from the previous line, so LastName's column type is "25S", FirstName's is "10", etc. RDB uses the letters S, N, and M to describe the three column types: string, numeric, and month. String is the default column type.

The integer value describes the column's width, but this is not an enforced limitation. Instead, the number is used by some of RDB's output formatting commands. Formatting is also the purpose of the ">" character in Phone's field type. The ">" and "<" indicate that a column should be displayed either right or left justified.

RDB data can contain any ASCII characters except for new-lines and tabs and there are no length limitations. The remaining lines of people.rdb are data lines, which are also tab separated. All RDB tables follow this basic format.

Using RDB

Another difference between RDB and traditional relational databases is that it uses a set of command-line programs to manipulate RDB data. Databases such as mySQL access data using Structured Query Language (SQL). SQL uses a SELECT statement to extract information from a data table. For example, if the information from the people.rdb table in Figure 1 was stored in a mySQL database table called people, a SELECT query might look like this:

```
SELECT FirstName, LastName, Phone
```

```
FROM people
WHERE LastName = "Smith"
```

This query tells mySQL to display three columns (FirstName, LastName, and Phone) from the people table, but only those rows of the table where the LastName field is equal to "Smith". The resulting table would look like this:

```
FirstName  LastName  Phone
---------  --------  --------
John       Smith     555-2416
Jane       Smith     555-3499
```

The same results are possible from RDB using two of its command-line programs and UNIX's pipe-and-filter mechanisms:

```
row LastName eq "Smith" < people.rdb | column
FirstName LastName Phone
```

RDB's **row** command is equivalent to the **WHERE** clause in SQL. The **column** command is similar to the column list that follows the **SELECT** keyword in the SQL example. It tells RDB which columns to display, and in what order. Notice that the Age field is not listed and therefore it won't appear in the output. The **row** command uses the following comparison operators: **gt**, **ge**, **lt**, **le**, **eq**, **ne**, **mat**, **nmat**. The last two operators are for regular expression matching. The keywords "and" and "or" can be used with the **row** command to build more complex queries. It's important also to note that each RDB command expects a valid RDB table as input, and emits a valid table as output. This allows complex database queries to be built by stringing together a collection of RDB commands. In addition to the **row** and **column** commands, RDB includes commands to join tables (**jointbl**), sort tables by a specific column or columns (**sorttbl**), and find unique rows by column (**uniqtbl**).

Other RDB commands, such as **mergetbl**, serve the same purpose that **INSERT**, **UPDATE**, and **DELETE** statements do in SQL, inserting, updating, or deleting rows from an RDB table. By leveraging the power of the UNIX command line, RDB makes it easy to perform complex data manipulations. Complex queries can be developed and debugged on the command line. These queries can then be saved as shell scripts. This simplifies extending RDB for a specific set of problems. **Joining Tables in RDB** All SQL-based databases allow you to join two tables on a column. RDB also lets you do this. The RDB table in Figure 2 has two columns -- Age and Comment. To join this table to the people.rdb table, use the **jointbl** command like this:

```
jointbl Age=Age ages.rdb < people.rdb
```

The first parameter indicates that we want to join the two tables on the Age column. The second parameter is the table to which we want to join. The first table, people.rdb, is expected from standard input. The result of the jointbl command is shown in Figure 3. Notice that the result is yet another RDB table, which means that the result of this join can be piped into subsequent RDB commands for further processing. The new table has five columns -- the four from people.rdb, and the Comment column from ages.rdb. But, where people.rdb had three data records, this table only has two because no age in ages.rdb matched the record for Jane Smith. The join only outputs records where the columns matched. There may be times when you want all the records from the first table to be output, even if they didn't match anything in the second table. This is a master-detail or outer join. To make an outer join with RDB, use **jointbl's -md** switch:

```
jointbl -md Age=Age ages.rdb < people.rdb
```

The table produced (Figure 4) has all three records from the first table, but the Comment column for Jane Smith's record is null. Notice the trailing tab character in the last record. **Formatting RDB Output** While RDB commands operate on simple text files, these tab-delimited files may not be the best way to present information to the user. RDB provides two commands that simplify producing reader-friendly reports. The first and simplest of these is the **ptbl** command. The **ptbl** command takes an RDB table from standard input and emits a fixed-column report as output. For example, running our example table through **ptbl**, like this:

```
ptbl < people.rdb
```

gives us output that looks like this:

```
LastName        FirstName   Age         Phone
--------------  ----------  ---  ------------
Smith           John         25     555-2416
Adams           Frank        27     555-1378
Smith           Jane         34     555-3499
```

Column widths and alignments are determined by the "column type" line of the RDB table. The **reporttbl** command allows more complex report layouts. It uses a separate form file that describes the report design. Figure 5 shows a sample form file, people1.frm, that could be used with the people.rdb table. The form file describes how to format each row from the RDB table. In this example, each table row will be output onto two lines, the first showing the person's name, and the second showing their age and phone number. Fields beginning with an ampersand character are picture fields, which describe the placement and alignment of individual columns. Following the ampersand will be one or more alignment characters -- "<" for left alignment or ">" for right

alignment. On the line immediately following the picture fields is a line listing the RDB column names to which each picture field corresponds. The form file is passed as a parameter to the **reporttbl** command, with the RDB table received via standard input:

```
reporttbl people1.frm < people.rdb
```

This command yields a report formatted like this:

```
Name: John       Smith
Age:    25       Phone: 555-2416

Name: Frank      Adams
Age:    27       Phone: 555-1378

Name: Jane       Smith
Age:    34       Phone: 555-3499
```

The reporttbl form file allows you to define optional page headers, as well as to embed the output from external commands such as **date** or **whois** into a report.

Uses for RDB

Programmers who write applications that generate log files can store those logs in RDB format, then use RDB to easily query those logs. RDB files are simply tab-delimited text files, so it's a simple matter to produce log files in RDB format. The hassles of parsing log files are reduced by breaking log files into RDB tables with columns like Date, Time, ApplicationName, Severity, and MessageText.

If you're writing CGI apps in either Perl or PHP, it's simple to read RDB tables and then display the data, which makes RDB easy to use for Web applications. I've used RDB extensively for archiving

data collected from electrical substations in near real time. Electrical utilities monitor large numbers of devices, such as breakers and transformers, and often need to save that data for later analysis. RDB works well in this role. Existing log files can easily be converted to RDB format. Once converted, these logs can be queried by date ranges, application name, message content, or any combinations of these fields.

Installing RDB

Installing RDB couldn't be much easier. Retrieve the tarball from the ftp site (**ftp.rand.org/pub/RDB-hobbs/RDB-2.6d.tar.gz**), unzip and untar it, **cd** into the rdb directory and type **make install**. A collection of Perl programs is installed, which means you need to have Perl installed on the target machine. Beyond that, there aren't really any prerequisites. By default, RDB tries to install to the **/usr/local/bin** directory, but this can be modified by editing the **INSTALL_DIR** line in the RDB **Makefile**.

There is one problem you may need to watch for: one of the RDB commands is called "column". This may conflict with the "column" utility included with many UNIX and Linux distributions (in particular, "column" is included with the util-linux package and installed with the Red Hat Linux distribution). You can avoid the problem by renaming one of the column utilities.

RDB doesn't have any daemon running, and there are no databases to create, so once it's installed, it's ready to use. All that remains is to create your RDB data tables, which you can do with a simple text editor.

Limitations of RDB

In practice, RDB works best on databases with predictable growth.

Log files grow by appending new records to the end of the file, which is one reason they work well as RDB tables. Likewise, data acquisition applications, where time-stamped data is continuously collected and archived, work well as RDB tables. Both these types of data files grow constantly, but it's seldom necessary to go back later and modify the records.

Also, because of the sequential nature of text file access, large files can be cumbersome within RDB. What constitutes "large" depends on the speed of the file system, the processor, and how long you're willing to wait for a query to complete. One advantage here is that once a file has been queried, subsequent RDB requests on the same file are likely to hit a cached copy of the file in memory. This can significantly cut down on query times. RDB has an **indextbl** command that can create a separate index file for an rdb table. This can also improve access times, but the index must be recreated each time the table is modified.

If a large amount of data is collected, disk storage may become a problem. Luckily, the UNIX philosophy ("small tools that do one thing well") helps solve this problem too. RDB tables can be compressed, using the gzip utility. This can reduce them to as little as a tenth their original size. When you need to query these files, you can simply access them with zcat, which uncompresses them and sends them to standard output.

Advantages of RDB

RDB tables are simple text files, so the whole UNIX command-line bag of tricks is available when using them. Collections of RDB files can be searched with grep. Output from RDB tables can be mailed using a command-line mail program. Repetitive sequences of RDB commands can be turned into shell scripts.

I use RDB to maintain a table of host names and IP addresses. The table also includes information about host name aliases, the physical location of the hosts, the purpose of the host (e.g., this host is used primarily as a Web server, and that host is a mail server), and the host's machine architecture (Intel, Alpha, etc.).

This data is used by several different programs, but primarily it's used to generate files used by our domain name servers.

My name server requires two files -- one that maps hosts to IP addresses, and another that handles reverse mappings from IP address to host name. I wrote a short Perl script that generates both these files from my hosts' RDB table. That way, I only have to update one file. The portability of RDB means that if I ever have to move the name server to a different machine, I can also move the hosts' RDB table with a minimum of fuss.

RDB can take advantage of the hierarchical directory structure of UNIX to speed data access and minimize file size. Date-stamped data can be stored in directories that indicate the date. For example, a log file for June 10th, 2001 can be stored in the directory /home/xyz/2001/06/10.

RDB data is trivial to exchange with other users. An RDB table can be emailed, intact, as an attachment and once received, it's ready to use. Because data is stored as simple ASCII text, it can move easily between different machines and different operating systems. RDB data files can be imported or pasted into spreadsheet applications such as Microsoft Excel or Star Office.

RDB is easy to share over the Web. You can store RDB tables in a Web directory, then access and query them remotely using a program like lynx. Lynx allows you to dump the source of a Web page (or in our, case an RDB table) to standard output, so lynx can

be used to quickly access data over the Internet. You can even join a remote table to a local RDB table.

RDB commands are built on the command line, and the ease of this approach lends itself to experimentation. If you're comfortable with the UNIX command line, and you routinely need flexible data analysis tools, give RDB a try. You can have the power of a relational database without giving up the flexibility of the UNIX command line.

Mark Pruett received a Masters Degree in Computer Science from Virginia Commonwealth University. He has been programming for the past 18 years. Mark currently works for Dominion Virginia Power. He can be reached at ***mpruett@mediaone.net****.*