

ESTTA Tracking number: **ESTTA411751**

Filing date: **05/30/2011**

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE  
BEFORE THE TRADEMARK TRIAL AND APPEAL BOARD

Proceeding	91193335
Party	Plaintiff Embarcadero Technologies, Inc.
Correspondence Address	MARTIN R GREENSTEIN TECHMARK A LAW CORPORATION 4820 HARWOOD RD, 2ND FLOOR SAN JOSE, CA 95124-5273 UNITED STATES MRG@TechMark.com, MPV@TechMark.com, AMR@TechMark.com, LZH@TechMark.com
Submission	Plaintiff's Notice of Reliance
Filer's Name	Leah Z Halpert
Filer's e-mail	MRG@TechMark.com, MPV@TechMark.com, LZH@TechMark.com, AMR@TechMark.com
Signature	/Leah Z Halpert/
Date	05/30/2011
Attachments	RSTUDIO-91193335-Embarcadero Notice of Reliance-Rebuttal Period.pdf ( 3 pages )(44060 bytes ) Exhibit B.pdf ( 12 pages )(414609 bytes ) Exhibit C.pdf ( 12 pages )(212616 bytes ) Exhibit D.pdf ( 9 pages )(51420 bytes ) Exhibit E.pdf ( 12 pages )(187170 bytes ) Exhibit F.pdf ( 15 pages )(1004103 bytes )

**IN THE UNITED STATES PATENT AND TRADEMARK OFFICE**  
**BEFORE THE TRADEMARK TRIAL AND APPEAL BOARD**

**EMBARCADERO TECHNOLOGIES, INC.**

**Opposer**

v.

**RSTUDIO, INC.**

**Applicant.**

**Opposition No.: 91-193,335**

**Trademarks: RSTUDIO**

**Serial Nos.: 77/691,980**

**77/691,984**

**77/691,987**

**OPPOSER'S NOTICE OF RELIANCE - REBUTTAL**

Pursuant to Trademark Rule 2.122(e) Opposer, EMBARCADERO TECHNOLOGIES, INC., (“Embarcadero”, or “Opposer”), by its attorneys, hereby gives notice that it will or may rely on the following materials relevant to the issues in the captioned proceeding, copies of which are attached to the Notice. All of the following materials are specifically entered into the record in order to rebut testimony and evidence submitted by Applicant during their testimony period.

The Notice of Reliance is being submitted prior to the close of Opposer’s Rebuttal Testimony Period, pursuant to the Trademark Trial and Appeal Board Manual of Procedure (TBMP) § 702.02, *See Sports Authority Michigan Inc. v. PC Authority Inc.*, 63 USPQ2d 1782, 1786 n.4 (TTAB 2001) (notices of reliance must be filed before closing date of party's testimony period).

1. RStudio Inc.’s current website as of May 25 2011, attached hereto as Exhibit A to show the manner in which Applicant has altered the site since their testimony period, and to show the manner in which Application is currently using and/or referencing to its RSTUDIO marks.
2. Working Paper No. 15 from the Meeting of the Management of Statistical Information Systems titled “R: An Open Source Statistical Environment Invited Paper” prepared by Valentin Todorov, UNIDO, dated March 28, 2008, available at:

- <http://www.unece.org/stats/documents/ece/ces/ge.50/2008/wp.15.e.pdf>, attached hereto as Exhibit B. The Working Paper shows the connectivity between the R computing language and relational databases as well as shows that statistical systems are never used in isolation, but rather must communicate with other systems.
3. The online book “Using R for Actuarial Science” by Shyamal Kumar, dated March 5, 2007, available at: [www.soa.org/files/pdf/UsingRforActuarialScience.pdf](http://www.soa.org/files/pdf/UsingRforActuarialScience.pdf), attached hereto as Exhibit C. The online book shows the connectivity between the R computing language and relational databases.
  4. The online article “Scenarios for Using R within a Relational Database Management System Server” by Duncan Temple Lang, dated April 12, 2001, available at: [www.omegahat.org/RSPostgres/Scenarios.pdf](http://www.omegahat.org/RSPostgres/Scenarios.pdf), attached hereto as Exhibit D. The article shows the connectivity between the R computing language and relational databases.
  5. Excerpts from the online manual “R Data Import/Export” edited by the R Development Core Team at the Comprehensive R Archive Network (CRAN), Version 2.13.0, dated April 13, 2011, available at: <http://cran.r-project.org/doc/manuals/R-data.pdf>, attached hereto as Exhibit E. This manual shows how the R language is well adapted to work with relational databases, such as ER/Studio.
  6. The article “Improving the analysis, storage and sharing of neuroimaging data using relational databases and distributing computing”, by Uri Hasson, et al, published in *NeuroImage*, a Journal of Brain Function, Volume 39, Issue 2, 15 January 2008, Pages 693-706, available at: [http://www.behaviormetrix.com/public\\_html/Hasson07.distrib.analysis.pdf](http://www.behaviormetrix.com/public_html/Hasson07.distrib.analysis.pdf), attached hereto as Exhibit F. This article shows how the R language is well suited to conduct analysis on relational databases

7. Portions of Embarcadero's current website as of May 6, 2011, as well as excerpts from various ER/Studio user guides, attached hereto as Exhibit G to show the relationship, interface, and/or interoperability between the ER/Studio product line and flat file or non-relational databases.

Dated: May 30, 2011

Respectfully Submitted,

EMBARCADERO TECHNOLOGIES, INC.  
By /Martin R. Greenstein/  
Martin R. Greenstein  
Mariela P. Vidolova  
Leah Z. Halpert  
TechMark a Law Corporation  
4820 Harwood Road, 2<sup>nd</sup> Floor  
San Jose, CA 95124-5273  
Tel: (408) 266-4700; Fax: (408) 850-1955  
E-Mail: MRG@TechMark.com  
Attorneys for Opposer

**CERTIFICATE OF SERVICE**

I hereby certify that a true and correct copy of the foregoing **OPPOSER'S NOTICE OF RELIANCE - REBUTTAL** is being served on May 30, 2011, by first class mail, postage prepaid on Applicant's Attorney of Record at his address below:

Charles E. Weinstein, Esq.  
Julia Huston  
Joshua S. Jarvis  
Anthony E. Rufo  
FOLEY HOAG LLP  
155 Seaport Blvd, Ste 1600  
Boston, MA 02210-2600  
Tel: (617) 832-1000  
E-Mail: [CEW@foleyhoag.com](mailto:CEW@foleyhoag.com)

/Leah Z Halpert/  
Leah Z Halpert

# Exhibit B

Distr.  
GENERAL

Working Paper No. 15  
28 March 2008

ENGLISH ONLY

**UNITED NATIONS STATISTICAL COMMISSION and  
ECONOMIC COMMISSION FOR EUROPE  
CONFERENCE OF EUROPEAN STATISTICIANS**

**EUROPEAN COMMISSION  
STATISTICAL OFFICE OF THE  
EUROPEAN COMMUNITIES (EUROSTAT)**

**ORGANISATION FOR ECONOMIC COOPERATION  
AND DEVELOPMENT (OECD)  
STATISTICS DIRECTORATE**

**Meeting on the Management of Statistical Information Systems (MSIS 2008)**  
(Luxembourg, 7-9 April 2008)

Topic (iii): Exchange/sharing/re-use of components, common models among statistical offices

### **R: An Open Source Statistical Environment**

#### **Invited Paper**

Prepared by Valentin Todorov, UNIDO

#### **I. INTRODUCTION**

1. The Open Source movement has changed dramatically the global software landscape in the recent decades. If we look at any software area we will find prominent representatives of the Open Source Software/Free Software (OSS/FS, also abbreviated as FLOSS/FOSS) like Linux, Apache, MySQL, Perl, PHP, OpenOffice, Mozilla Firefox. The exact definition of what Open Source is and what it is not can be found at the home page of the [Open Source Initiative](#), but briefly speaking, programs developed as OSS/FS are programs with a licenses giving the users the freedom to redistribute them in any form, to use them for any purpose, to have access to the complete source and have the freedom to modify it and to redistribute the modified programs, of course without having to pay any royalties to the original developers. An extensive quantitative evaluation of the Open Source approach can be found in Wheeler (2007).
2. In the world of commercial statistical software there are only a few very well known names that dominate - SAS, SPSS, STATA, S-PLUS, MATLAB. The situation with the free software is similar. Although there are hundreds and hundreds of free tools for solving a given statistical problems, if we talk about a comprehensive statistical environment which could be competitive to the dominating commercial packages, the choice is not much and we end always with R.
3. The goal of this paper is to present a brief overview of the Open Source statistical language and environment R, pointing out its advantages (and disadvantages) when compared to the commercial statistical packages dominating the market for statistical software. Since nowadays it is very easy to find any information (useful or not) in Internet, not many references are included and the included ones are either those that I have used for the preparation of this material or such that are not easy to find.

## II. WHAT IS R

### A. The R Platform

3. As described by the R-core development team on its web page, R is “a system for statistical computation and graphics. It provides, among other things, a programming language, high-level graphics, interfaces to other languages and debugging facilities.”
4. R is a GNU project, which was developed after and can be considered a different implementation of the S language and environment, with similar syntax and features. The S language was developed at Bell Laboratories (formerly AT&T, now Lucent Technologies) by John Chambers and colleagues in the mid-seventies. One of the big names in the world of the commercial statistical software is S-Plus, which is a value added implementation of the S language and now is marketed by Insightful Corporation. Despite the very close similarities with S and the superficial similarities with the C language, actually the R engine is significantly influenced by Scheme, a Lisp dialect. Nevertheless, much code written for S runs unaltered under R.
5. The development of the R language and environment first started in 1990 as an experimental project by Ihaka and Gentleman, both from the laboratory of statistics at the University of Auckland (New Zealand), in 1993 a preliminary version of R was presented and already in 1995 R was released under the GNU Public License. Now the development of R is managed by the R-core team consisting of 17 members including John Chambers.
6. R provides a wide variety of statistical (linear and non-linear modelling, classical statistical tests, time-series analysis, classification, clustering, robust methods and many more) and graphical techniques.
7. A general collection of useful information for users on **all** platforms (Linux, Mac, Unix, Windows) can be found in [R FAQ](#). Additionally there are two platform-specific FAQs for Windows and MacOS.

### B. R Availability

8. R is available as Free Software under the terms of the [Free Software Foundation's GNU General Public License](#) in source code form.
9. R can be obtained as both source and binary (executable) forms from the [Comprehensive R Archive Network \(CRAN\)](#). The source files are available for a wide variety of UNIX platforms and similar systems (including FreeBSD and Linux) as well as for Windows and MacOS for which are available also precompiled binary distributions of the base system and contributed packages.
10. The most recent release of R is version 2.6.2 (released on 8.February 2008) and pre-release versions of 2.7.0 are in progress.
11. A wide variety of add-on functionality (actually the normal way of extending R) is available from the same web page in the form of contributed R packages, which can be downloaded in source form or installed directly from the R console by using the *install.packages()* function (provided the computer is connected to the Internet).

### C. Is R harder to learn/use than other statistical packages?

12. One of most popular criticisms against R is that this statistical language is hard to learn, compared to the other statistical packages, like SAS and SPSS and it is said to have a very steep learning curve. Quoting Kabacoff (2008) ”I have been a hardcore SAS and SPSS programmer

for more than 25 years, a Systat programmer for 15 years and a Stata programmer for 2 years. But when I started learning R recently, I found it frustratingly difficult”. On the other hand, I have not used SAS for 25 years, but I have programmed in C as long, I have not used Systat or Stata for many years, but I have programmed in Fortran, Java, C# and many other (non statistical) programming languages. And for me R was not harder to learn than any of these (non statistical) languages.

13. As mentioned in Muenchen (2007), while SAS and SPSS have a wide variety of functions and procedures, all these fall into one of five categories and these are:
  - (a) Data input and management statements for reading, transforming and organizing the data
  - (b) Statistical and graphical procedures for analysing the data
  - (c) An output management system for formatting the output from statistical procedures or for customizing printed output. In SAS this is done by the Output Delivery System (ODS) while in SPSS it is done by the Output Management System (OMS).
  - (d) A macro language to allow creating of programs, i.e. repeatedly executing statements, functions and procedures
  - (e) A matrix language for creating new algorithms. This language is SAS/IML in SAS and SPSS Matrix in SPSS.
  
14. In SAS and SPSS these five areas are handled with different systems, but for the sake of simplicity the introductory training in these packages involves mainly the first two (data management, statistical analysis and graphics) and many of the users stay with this knowledge and never learn the more advanced topics. On the other hand, in R all these five areas are interrelated in such a way that the user must approach them in parallel, which could be difficult for the novice. But the integration of these five areas gives R a significant advantage in power which allowed most of the R procedures to be written in the same interpreted language and thus the source code of these procedures is available for viewing and modifying by the user.

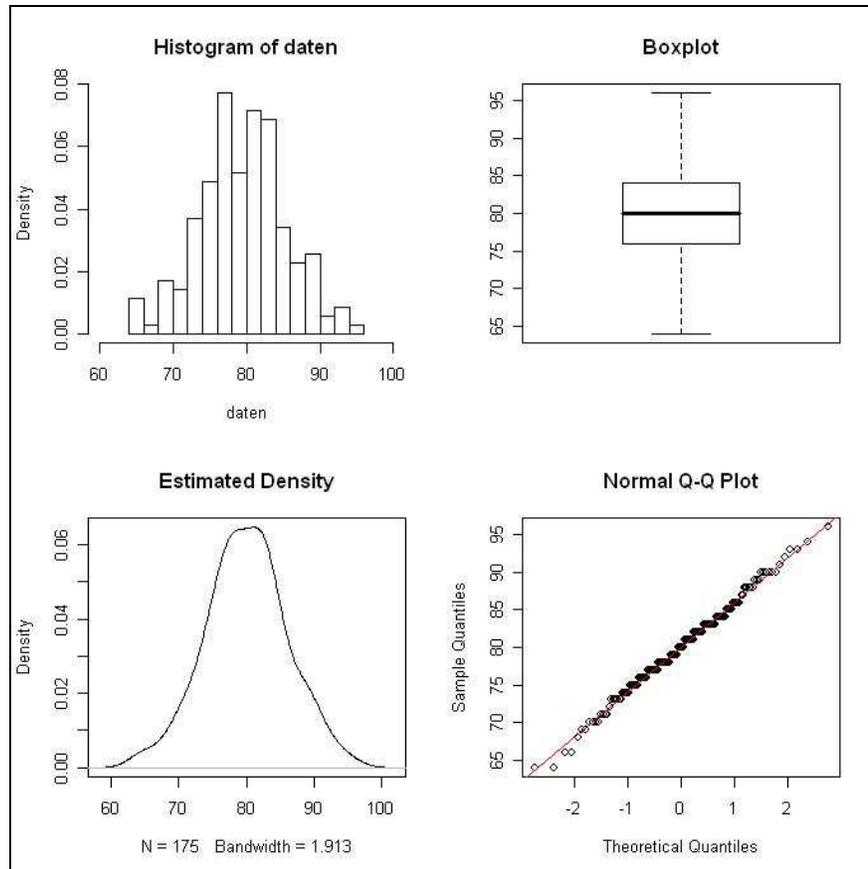
#### D. R Graphics

15. One of the most important strengths of R is the ease with which simple exploratory graphics as well as well-designed publication-quality plots can be produced, including mathematical symbols and formulae where needed. Great care has been taken over the defaults for the minor design choices in graphics, but the user retains full control.
  
16. A simple example of basic R graphics is shown in *Figure 1* (produced by the code below) while *Figure 2* shows a trellis-type graphic (to my knowledge this type of graphics are not available in most of the other statistical packages). Finally, *Figure 3* shows an example of time series diagnostic graphics.

```

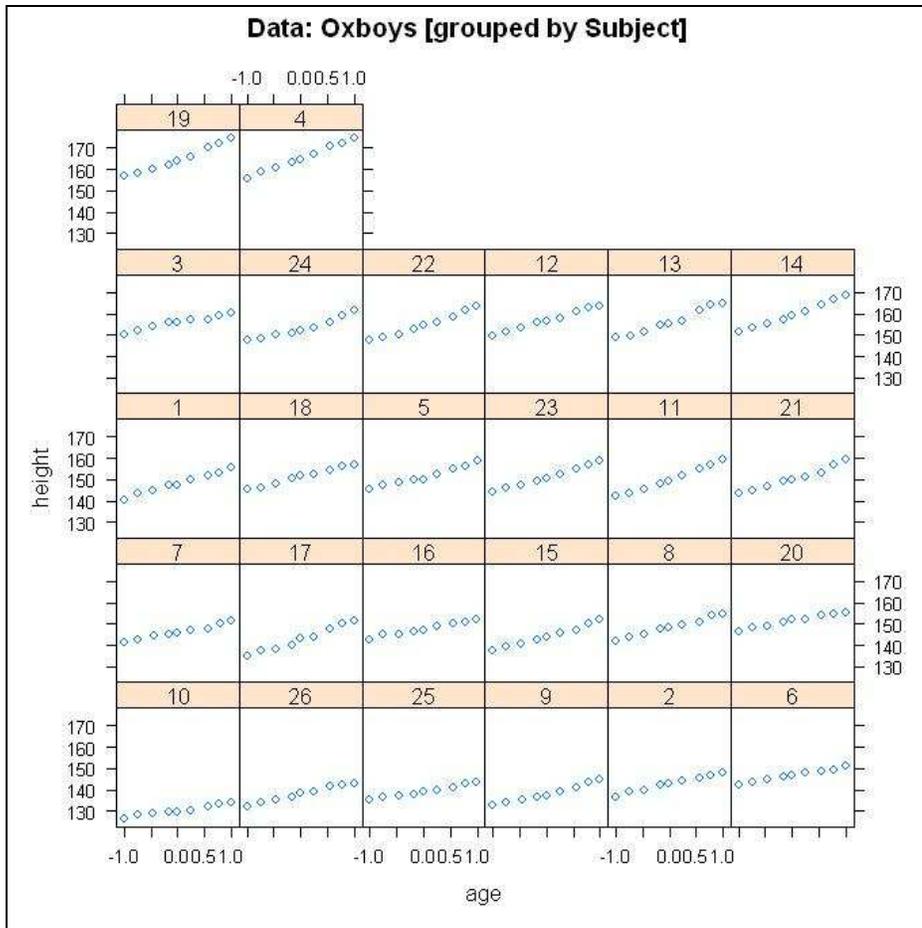
daten <- faithful[faithful$eruptions > 3, 2]
par(mfrow = c(2, 2))
hist(daten, freq = FALSE, breaks = 20, xlim = c(60,100))
boxplot(daten, main = "Boxplot")
plot(density(daten), main = "Estimated Density")
qqnorm(daten)
qqline(daten, col = "red")

```



**Figure 1:** A simple example of R basic graphics

17. R can produce graphics in many formats, including:
  - (a) On screen
  - (b) PDF files for including in LATEX or for direct distribution
  - (c) PNG or JPEG bitmap formats for the WEB
  - (d) On Windows, metafiles for Word, PowerPoint, and similar programs
  
18. An exciting example of the R graphics capabilities is the [R Graph Gallery](#) which aims to present many different graphics fully created with the programming environment R. Graphs are gathered in a MySQL database and browsable through PHP. An excellent reference to R Graphics is the book of Paul Murrell, a member of the R Core Development Team who has not only been the main author of the *grid* package but has also been responsible for several recent enhancements to the underlying R graphics engine.



*Figure 2: An example of multipanel graphic display (trellis graphics) in R*

### **E. R Extensibility (R Packages)**

19. One of the most important features of the R language and one of the main topics in which R beats the commercial version S-Plus, is its extensibility by creating packages of functions and data. This was one of the key features that has contributed to the R's growth. The first step in R programming is writing R functions for performing a given repetitive action. Next, to facilitate the reusability of such functions they can be combined into a package.
20. The R package mechanism was first designed to help the developers to encapsulate related programs, data and documentation and distribute them to the users. Now this mechanism is the natural way of extending R. Numerous researchers create R packages and post them on the special area of the Comprehensive R Archive Network (CRAN). The R environment provides tools for downloading and installing packages, for creating packages from scratch and extending them, for writing and incorporating online help pages as well as extended documentation in PDF format. A package can contain R code, documentation files in a special format which provide both online help as well as printed manual and example data sets, but could contain also compiled C or Fortran source code which is automatically compiled when building the package.
21. The package 'check' tool is invaluable for the package developer and a positive result from the check is a must for posting a package on CRAN. Apart from the formal code validation, the check procedure includes running all examples from every help page and will build and test the package vignettes, if available. It is possible to write test cases in the form of R programs, which

will be run, and the output will be compared with the previously stored output. This guarantees that no side effects, which broke already working code, appeared with the last changes to the package.

22. Currently more than 1300 packages exist on CRAN covering a wide variety of statistical methods and algorithms, including the newest achievements in the statistical science. There are about a dozen ‘base’ packages, which together with the packages denoted as ‘recommended’ are included in all binary distributions of R.

#### **F. R and the Others (Interfaces)**

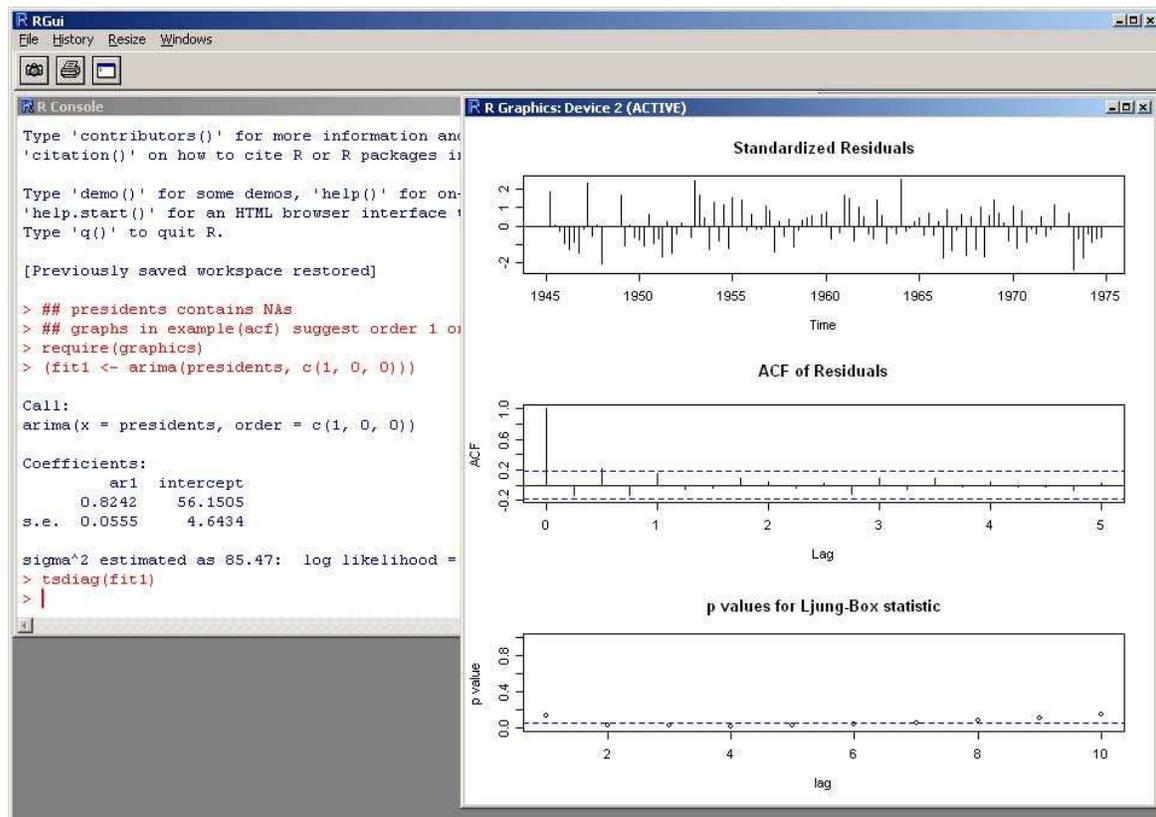
23. When using a statistical system we must have in mind that this is not done in isolation and the system must be able to communicate with other systems in order to import data for analysis, to export data for further processing (use the right tool for the right work) and to export results for report writing.
24. A rich variety of facilities for data import and export as well as for communication with databases, other statistical systems and programming languages are available either in R itself or through packages available from CRAN.
- (a) The easiest data format to import into R is a simple text file but reading XML, spreadsheet like data, e.g. from Excel is also possible;
  - (b) The recommended package `foreign` provides import facilities for reading data in the format of the statistical packages Minitab, SAS, S-Plus, SPSS, STATA, Systat and Octave as well as export capability for writing STATA files, while the package `matlab` provides emulation for Matlab;
  - (c) Working with large data sets could be a problem in R (if the data do not fit in the RAM of the computer) but the interface to RDBMS could help in such cases. Another limitation is that R does not easily support concurrent access to data, i.e. 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. This could also be solved by using the interface to relational databases. There are several packages available on CRAN for communication with RDBMSs, providing different levels of abstraction. All have functions to select data within the database via SQL queries, and to retrieve the result as a whole, as a data frame or in pieces (usually as groups of rows). Most packages are tied to a particular database – ROracle, RMySQL, RSQLite, RmSQL, RPgSQL, while the package RODBC provides a generic access to any ODBC capable relational database.
  - (d) R is an interpreted language and some very computation intensive algorithms could be slow – in this case a native code implemented in C or FORTRAN is the right solution;
  - (e) R has no real (statistical) GUI, which is often criticized by the proponents of point-and-click statistical packages, but if it is necessary to develop a nice specialized graphical user interface, this could be implemented in Java and R will do the computations in the background

#### **G. R for Time Series**

25. R has extensive facilities for analysing time series in the packages `stats`, `tseries`, `zoo`, `its` (irregular time series), `ast` (not yet on CRAN), `pastecs` (for analysing space-time ecological time series) and `lmtest`. Vito Ricci has compiled a reference card of the most popular time series functions – see Ricci (2008). The package `stats` includes classical time series modelling tools - `arima()` for ARIMA modelling and Box-Jenkins-type analysis. For fitting structural time series is available `StructTS()` in `stats` and for time series filtering and decomposition can be used `decompose()` and `HoltWinters()`. The package `forecast` supplements the tools available in the `stats` package by providing additional forecast methods, and graphical tools for displaying and

analysing the forecasts. Further information on time series functions and packages in R can be found in the Task View Econometrics - <http://cran.r-project.org/web/views/Econometrics.html>.

26. The functionality for analysing monthly or lower frequency time series data which is implemented in the software packages **TRAMO/SEATS** (Time series Regression with ARIMA Noise, Missing values and Outliers/Signal Extraction in ARIMA Time Series) and **X-12-ARIMA** seasonal adjustment software of the US Census Bureau is easily accessible through the *Gretl* library – see Cottrell (2008).
27. In *Figure 3* is shown an example of time series analysis functions - *arima()* for fitting an ARIMA model to a univariate time series and *tsdiag()* for plotting time series analysis diagnostics



*Figure 3: Example of time series analysis functions: arima() for fitting an ARIMA model to a univariate time series and tsdiag() for plotting time series analysis diagnostics*

## H. R for Survey Analysis

28. Complex survey samples are usually analysed by specialized software packages. From the most well known general-purpose statistical packages Stata provides much more comprehensive support for analysing survey data than SAS and SPSS and could successfully compete with the specialized packages. In R functionality for survey analysis is offered by several add-on packages, the most popular being the *survey* package. Detailed information can be found in the manuals of the package as well as from its home page, maintained by the author, Thomas Lumley at <http://faculty.washington.edu/tlumley/survey/>, but here is a brief overview:
  - (a) Designs incorporating stratification, clustering, and possibly multistage sampling, allowing unequal sampling probabilities or weights; multistage stratified random sampling with or without replacements

- (b) Summary statistics: means, totals, ratios, quantiles, contingency tables, regression models, for the whole sample and for domains
  - (c) Variances by Taylor liberalization or by replicate weights (BRR, jack-knife, bootstrap, or user-supplied)
  - (d) Post-stratification and raking
  - (e) Graphics: histograms, hexbin scatterplots, smoothers.
29. Other relevant R packages are *pps*, *sampling*, *sampfling*, all of which focus on design, in particular PPS sampling without replacement.

### I. R and SDMX

30. No, there is nothing of the kind but this is not much different from the other statistical software packages. There exists a proposal for a new data exchange format for statistical data based on XML, which is called StatDataML – see Meyer (2004) – and a corresponding implementation in the R package *StatDataML*.

### J. R and the Outliers (Robust Statistics in R)

31. Atypical observations, which are inconsistent with the rest of the data or deviate from the postulated model, usually called outliers, are likely to appear often in the data sets under consideration. Unfortunately most of the classical statistical methods are very sensitive to such data. Therefore *robust* statistical methods are developed whose main goal is to produce reasonable results even when one or more outliers may appear anywhere in the data.
32. As an example let us consider the “Wages and Hours” data set available at <http://lib.stat.cmu.edu/DASL/>. The data are from a national sample of 6000 households with a male head earning less than \$15,000 annually in 1966. The data were classified into 39 demographic groups (if the cases with missing data are removed only 28 groups remain). The study was undertaken to estimate the labour supply (average hours) from the available data. There are 9 independent variables but for the sake of the example we will consider only one - the average age of the respondents, i.e. if  $y$  = labour supply and  $x$  = average age of the respondents we will fit the model  $y = \beta_0 + \beta_1 x$ .
33. **Figure 4** (a) shows a scatterplot of the data with the line fitted by the Ordinary Least Squares (OLS) method. It is clearly seen that two observations are outliers – assuming that the measurements are correct, the average age of the people in group 3 is very low compared to the others while the age of the people in group 4 is on average too high. The line shown in (a) does not fit well the data since it is attracted by the outliers. The outliers fall inside the tolerance band in the residual plot presented in (b). The line fitted by the robust Least Trimmed Squares (LTS) method, presented in (c) resists the outliers and follows the majority of the data. The corresponding residual plot shown in (d) clearly identifies the outliers. For further examples based on this data set as well as other information and references about robust statistics see Hubert et al. (2004).
34. SAS and Stata have functions for robust regression (*PROC ROBUSTREG* and *rreg* respectively) while SPSS has no such capability. In R robust methods are available in many packages, the most well known being *MASS*, *robustbase*, *rrcov*, *robust*. The computation and the diagnostic plots shown in **Figure 4** were produced by the function *ltsReg()* from package *robustbase*.
35. A method for visualization of missing data and robust imputation is developed by Templ et al. (2008) and implemented in R (not yet on CRAN)

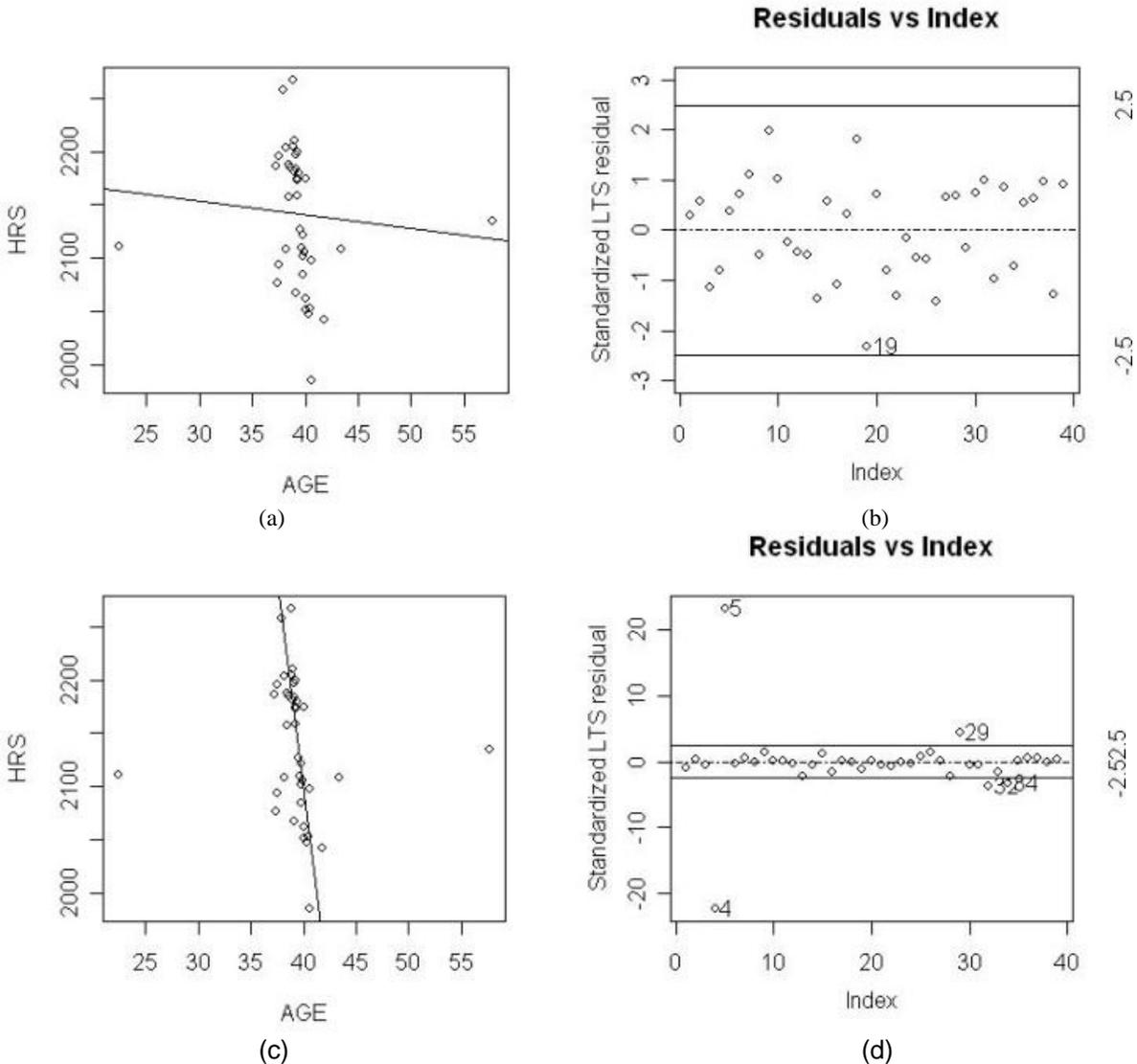


Figure 4: Wages and Hours data: Scatter plot of the data with LS (a) and LTS (c) line as well as their residual plots – (b) and (d) respectively

## K. More R

36. There are many more topics about R, which deserve our attention but were not considered here because of time and space limitations. A brief list is in order.
- R and the WEB** – there are several projects that provide possibility to use R as a service over the Web. The first one was Rweb – see [Rweb Home Page](#), which provides three types of interface: a simple text version, a more sophisticated JavaScript based interface and a point-and-click interface, mainly suitable for teaching statistics. Information about other Web projects for R is available from [R FAQ](#).
  - R and the Missing** - Missing values in both character and numeric variables are represented by the symbol NA (not available) and most modelling functions offer options for dealing with missing values. Advanced handling of missing values is available through a number of packages:
    - mvnmle*: ML estimation for multivariate normal data with missing values;
    - mitools*: Tools for multiple imputation of missing data and to perform analyses and combine results from multiple-imputation datasets;

- iii. *mice* - Multivariate Imputation by Chained Equations;
  - iv. *EMV*: Estimation of Missing Values for a Data Matrix
- (c) **R GUI.** As already mentioned, R has no (statistical) GUI and this is a reason often to be criticized by the proponents of the point-and-click statistical packages. Nevertheless also statistical GUIs are emerging and already several packages exist like *Rcomander* and *Sciviews*.
- (d) **R Objects.** One of the main features of R is that it is an object oriented (although not in the sense C++, Java and C# are) language.

### III. SUMMARY

37. In the following *Table 1* is presented a summary of the main differences between R and SAS and SPSS. The second part of the table is a comparison of the SAS and SPSS products to the approximate equivalent R packages. The table was compiled mainly from Muenchen (2007) but only the topics, which are of interest for our work, are considered.

Topic	SAS	SPSS	R
Output Management System	Rarely used for routine work		Output is easily passed from one function to another to do further processing and obtain more results
Macro language	A special language used for performing repetitive tasks and adding new functionality. The new functions are not run in the same way as the built-in procedures		R itself is a programming language. The added new functions are run exactly in the same way as the built-in ones.
Matrix language	A special language used for adding new functionality. The new functions are not run in the same way as the built-in procedures		The base R itself is a vector- and matrix-based language, and it ships with many powerful tools for doing matrix manipulations. These are complemented by the packages Matrix and SparseM.
Publishing results	Cut and paste to a Word processor or exporting to a file		There are possibilities to produce Tex output (including graphics) using the Sweave package
Data size	Limited by the size of the disk		Limited by the size of the RAM, (not trivial) usage of databases for large data sets is possible
Data structure	Rectangular data set		Rectangular data frame, vector, list
Interface to other programming languages	Not available		R can be easily mixed with Fortran, C, C++ and Java

Source code	Not available		The source code of R as well as of the R packages is a part of the distribution
Basics	SAS®	SPSS Base™	R
Data Access	SAS/ACCESS®	SPSS Data Access Pack™	DBI, RODBC, foreign
Data Mining	Enterprise Miner™	Clementine®	rattle, arules, FactoMineR
Geographic Information Systems /Mapping	SAS/GIS®, SAS/Graph®	SPSS Maps™ (no full GIS)	maps, mapdata, mapproj, GRASS via spgrass6, RColorBrewer, see Spatial in Task Views
GUI	Enterprise Guide®	SPSS Base™	JGR, R Commander, pmg, Sciviews
Graphics	SAS/GRAPH®	SPSS Base™	ggplot, gplots, grid, lotrix, graphics, gridBase, hexbin, lattice, vcd, vioplot, scatterplot3d, geneploader, Rgraphics,
Dynamic Graphics	SAS/INSIGHT®	None	GGobi via rggobi iPlots, Mondrian via Rserve
Matrix/Linear Algebra	SAS/IML®	SPSS Matrix™	R, matlab, Matrix, sparseM
Missing Values Imputation	SAS/STAT®: MI	SPSS Missing Values Analysis™	aregImpute (Hmisc), EMV, fit.mult.impute (Design), mice, mitools, mvnmle
Sampling, Complex or Survey	SAS/STAT®: surveymeans, etc.	SPSS Complex Samples™	pps, samplng, sampling, spsurvey, survey
Time Series	SAS/ETS®	SPSS Trends™ Expert Modeler	Many (> 40) packages - see Task View Econometrics.

*Table 1: Functionality of R, SAS and SPSS*

#### IV. REFERENCES

- Banfield, J. (1999) Rweb: Web-based Statistical Analysis, *Journal of Statistical Software*, **4**, 1
- Cottrell, A. (2008) Gnu Regression, Econometrics and Time-series Library, URL: <http://gretl.sourceforge.net/index.html>
- Grunsky, E.C., (2002) R: a data analysis and statistical programming environment—an emerging tool for the geosciences, *Computers & Geosciences*, **28** 10, pp 1219-1222.
- Hornik, K and Leisch F, (2005) R Version 2.1.0, *Computational Statistics*, **20** 2 pp 197-202
- Kabacoff, R. (2008) Quick-R for SAS and SPSS users, available from <http://www.statmethods.net/index.html>
- López-de-Lacalle, J, (2006) The R-computing language: Potential for Asian economists, *Journal of Asian Economics*, **17** 6, pp 1066-1081
- Meyer, D, Leisch, F., Hothorn, T. and Hornik, K., StatDataML: An XML Format for Statistical Data, URL: <http://citeseer.ist.psu.edu/546737.html>
- Muenchen, R. (2007), R for SAS and SPSS users, URL: <http://oit.utk.edu/scc/RforSAS&SPSSusers.pdf>
- Muenchen, R. (2007), Comparison of SAS and SPSS Products with R Packages and Functions, <mailto:BobM@utk.edu>
- Murrel, P. (2005) R Graphics, Chapman & Hall
- R Development Core Team (2007) R: A Language and Environment for Statistical Computing, R Foundation for Statistical Computing, Vienna, Austria, ISBN 3-900051-07-0. URL: <http://www.r-project.org/>
- Templ, M and Filzmoser, F (2008), Visualisation of Missing Values and Robust Imputation in Environmental Surveys, submitted for publication
- Vito R. (2008) R Functions for Time Series, URL: <http://cran.r-project.org/doc/contrib/Ricci-refcard-ts.pdf>
- Wheeler, D.A., (2007) Why Open Source Software / Free Software (OSS/FS, FLOSS, or FOSS)? Look at the Numbers! URL: [http://www.dwheeler.com/oss\\_fs\\_why.html](http://www.dwheeler.com/oss_fs_why.html)

# Exhibit C

# Using R for Actuarial Science

1	Introduction	2
2	Some Features of R	2
3	SOA Tables Database	4
3.1	Binary Files of SOA DB	4
3.2	Implementation	6
4	Vectorization	9
5	Conclusion	11

Shyamal Kumar

# 1 Introduction

The prestigious 1998 Association for Computing Machinery Award for software systems was awarded to John Chambers of Bell Labs for the S system. The [citation](#) reads, *For the S system, which has forever altered how people analyze, visualize, and manipulate data.* R is an open source implementation of S and S-PLUS its commercial implementation.

R is an extensible, well documented language and environment with a core group of developers spread out over many countries. The size of its global user group and the diversity in its applications are some of its many strengths. Aimed at actuaries and especially actuarial science students who are pondering over the choice of a computing platform to complement MS Excel, the insurance industry defacto standard, this article makes a case for R.

Designed for statistical computing and embraced by scores of statisticians, most if not all statistical needs of an actuary should be ably served. Hence discussion of its statistical prowess will be conspicuously absent. The next section will highlight some features of potential interest to actuaries. Life contingent computations use mortality tables and most of the important tables are found in the SOA tables database. The third section discusses an implementation of access to this binary database with the intention of providing the readers with a good starting point for computing on the life side. Vectorization, the subject of the penultimate section, is an important concept for computing on R, like on APL and other interpreted vector languages . There we discuss a vectorized solution for an important class of actuarial algorithms.

## 2 Some Features of R

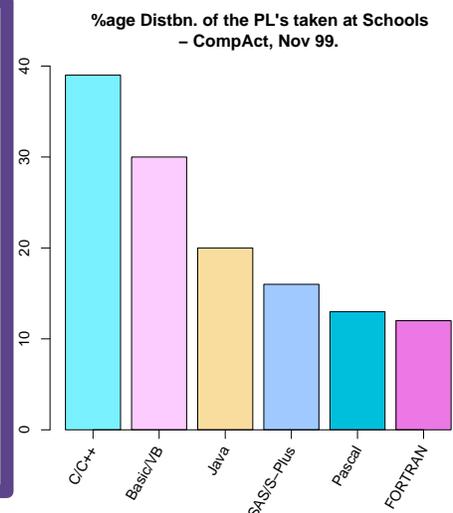
First, R has an effective programming language with a simple syntax. [The R Language Definition](#), describes the syntax as having *"a superficial similarity with C, but the semantics are of the FPL (functional programming language) variety with stronger affinities with Lisp and APL."* [A Brief History of S](#) by Richard A. Becker is a worth a read to know of the influences of other languages on the development of S. For example, the following is taken from it; *"... the basic interactive operation of S, the parse/eval/print loop, was a well-explored concept, occurring in APL, Troll, LISP, and PPL, among others. From APL we borrowed the concept of the multi-way array (although we did not make it our basic data structure), and the overall consistency of operations. The notion of a typeless language (with no declarations) was also present in APL and PPL."*

Second, R is highly extensible and seamless extensions are carried out using packages. Importantly, C and FORTRAN code can be linked and called at run time from R not only enabling

use of existing code but also providing avenues for accelerating computationally intensive tasks. And in the other direction, R objects can be manipulated in C. Talking about speed, support for BLAS (Basic Linear Algebra Subroutines) is provided in R. Also, there is a package called snow which implements a simple mechanism for computing on workstation clusters. For more details consult [Writing R Extensions](#).

Third, its reputed ability to produce high quality graphics with a minimal amount of code. The base graphics system contains both high and low level commands. For a demo of its capabilities, type `demo(graphics)` on the R command prompt. Especially important is its ability to generate graphics for publication in many formats including postscript, pdf, jpeg and png. Also, there is a separate graphics sub-system called grid which is considered to be more powerful and another package called gridBase for combining grid and base graphics output. Below is an example of the use of gridBase, similar to one in [Murrell \(2003\)](#), to combine the ease of a highlevel base graphics system command to draw a barplot with the power of grid used here to rotate the x-axis labels.

```
midpts <- barplot(c(39,30,20,16,13,12), axes = FALSE,
  col=c("#7AF1FE", "#FCCBFF", "#F8DD9E", "#9FC9FF",
  "#00BFDD", "#EB78E4"), ylim=c(0,40));
axis(2); axis(1, at = midpts, labels = FALSE);
vps<-baseViewports();
pushViewport(vps$inner, vps$figure,vps$plot);
grid.text(c("C/C++", "Basic/VB", "Java", "SAS/S-Plus",
  "Pascal", "FORTRAN"), x=unit(midpts, "native"),
  y=unit(-1, "lines"), just="right", rot=60);
popViewport(3);
```



And last but not the least, R has many utilities for accessing data. Access to data resident in a Relational Database Management Systems is provided by several packages. Any system providing an ODBC interface can be accessed using RODBC. This list not only includes most important RDBMSs but also databases like MS Access. Also, on Windows, ODBC drivers for text files, Excel files and Dbase files are available. System specific packages are available for Oracle and MySQL. R provides tools for accessing data in a binary format which is used in the implementation of the next section. There is a package to parse XML files using either the DOM or the SAX mechanism. Of course, there is a rich support for reading text files. Also, there are packages which make R a DCOM client (and server too) which help in accessing for example, live (financial) data. [R Data Import/Export](#) is the relevant manual for details.

Since R is well documented, the above description has been kept brief. Moreover, the above list is just a small subset of its wide ranging capabilities. Observe that some of the described

features are not part of the *core* of R but of one of the many packages which seamlessly extend it. Access to such features requires installation of relevant package(s), a matter of just a few *clicks*. At this point, a tour of its official web site at <http://www.r-project.org> and a browse of [An Introduction to R](#) would be a good idea. And if already convinced to try it out, go ahead and install your free download!

## 3 SOA Tables Database

Here we discuss an implementation of access to the SOA tables database. To keep the discussion self contained, the first sub-section describes the SOA tables database files (binary format) [tables.dat](#) and [tables.ndx](#). The next sub-section discusses the implementation along with some examples of its use.

### 3.1 Binary Files of SOA DB

The database owes its existence to a joint project of the Technology Section of the SOA, the International Section of the SOA and other volunteers. It consists of two binary files - `tables.dat`, the data file, and `tables.ndx`, the index file. The description of the storage format for these files can be found in the help of [TableManager](#), a software copyrighted by Steve Strommen, FSA, MAAA, and distributed freely by the SOA. The description is repeated below to keep the article self contained. The [Table Manager web page](#) also has a MS Excel addin available for free download for those interested in importing table values directly into Excel. A visit to the page is highly recommended.

#### `tables.ndx`

The primary purpose of this file, as any index file, is to facilitate fast access to a table from the `tables.dat` file. This is done by providing the offset, i.e. the number of bytes from the start of the file, at which the data for a table begins. This is a binary file with a sequence of fixed size records with an initial offset of 58 bytes (due to descriptive information). Each record has five fields as depicted in the figure below. The character strings in this file, unlike those of the `tables.dat`, are NULL byte terminated (as in C). The usage code is the same as that of the data file.

4 Bytes	50 Bytes	4 Bytes	31 Bytes	1 Byte
32-bit Integer	Character String	32-bit Integer	Character String	8-bit Integer
Table Number	Table Name	Offset	Country	Usage Code
4	54	58	89	90

**Figure 1** An Index File Record

## Record Types for tables.dat

Type	Content	Storage Format	Possible Values
1	Table Name	Character String	
2	Table Number	32-bit Integer	
3	Table Type	Single Character	<ul style="list-style-type: none"> <li>• S - Select</li> <li>• A - Aggregate by Age</li> <li>• D - Aggregate by Duration</li> </ul>
4	Contributor	Character String	
5	Source	Character String	
6	Volume	Character String	
7	Observation Period	Character String	
8	Unit of Observation	Character String	
9	Construction Method	Character String	
10	Reference	Character String	
11	Comments	Character String	
12	Minimum Age	16-bit Integer	
13	Maximum Age	16-bit Integer	
14	Select Period	16-bit Integer	
15	Maximum Select Age	16-bit Integer	
16	No. of Decimal Places	16-bit Integer	
17	Table Values	Sequence of 8-byte IEEE floating types	
18	Hash Value	32-bit Unsigned Integer	
19	Country	Character String	
20	Usage	8-bit Integer	<ul style="list-style-type: none"> <li>• 0 - No Data</li> <li>• 1 - Insured Mortality</li> <li>• 2 - Annuitant Mortality</li> <li>• 3 - Population Mortality</li> <li>• 4 - Selection Factors</li> <li>• 5 - Projection Scale</li> <li>• 6 - Lapse Rates</li> <li>• 99 - Miscellaneous</li> </ul>

### tables.dat

This binary file is a sequence of generic records with a single consecutive sub-sequence of such records (terminating with a record of type 9999 with missing length and data fields) pertaining to a single table. The storage format of a generic record is shown below.

Record Type Code - 16 bit Integer	Length of the Data - 16 bit Integer	Data - Variable Length
-----------------------------------	-------------------------------------	------------------------

**Figure 2** A Generic Record

There are twenty record types as listed in the following inset. A caveat is that not all the record types are relevant to a table and moreover not all the relevant types are necessarily present. The latter becomes important as the data packaged as list may contain some *empty* components. The character strings neither have a terminating NULL byte (like C) nor a leading byte containing its length (like Pascal) - but note that the length can be read off the third and fourth bytes of the generic record. R as a platform is convenient to read binary files as it has support for reading IEEE floating point numbers.

The table values are stored in the ascending order of the index (Age or Duration) for types "A" and "D". For type "S", the order is issue age wise in the ascending order of the select duration until we hit the maximum issue age and thereafter in the ascending order of the age.

## 3.2 Implementation

### RCode for TblSearch

```
"TblSearch" <-
function (Na = "", C = "", U = "", No = "")
{
  rec <- function(...) {
    c(readBin(z, integer(), size = 4), sub("", "", readChar(z,
      50)), readBin(z, integer(), size = 4), sub("", "",
      readChar(z, 31)), readBin(z, integer(), size = 1))
  }
  readChar(z <- file("tables.ndx", "rb"), 58)
  x <- sapply(1:((file.info("tables.ndx")$size - 58)/90), rec)
  close(z)
  apply(matrix(c(Na, C, U, No, 2, 4, 5, 1), 4, 2), 1, function(y) {
    x <<- x[, grep(y[1], x[as.integer(y[2]), ], perl = TRUE),
      drop = FALSE]
  })
  data.frame(No = as.integer(x[1, ]), Name = x[2, ], Country = x[4,
    ], Usage = as.integer(x[5, ]), Offset = as.integer(x[3,
    ]))
}
```

The first function, `TblSearch`, provides a query facility for the file `tables.ndx`. The query has to be in the form of a [PERL regular expression](#) and can be over each of the fields excepting offset. The result is the set of records satisfying all of the individual constraints (intersection) packaged as a dataframe object. This function not only helps a user search for tables in the database but also is used by the function `Tbl`. The following are some comments on the code.

- i. A user may leave unspecified the trailing arguments which would then assume the default null value..
- ii. The function `rec` reads the five fields for each record and is used by the `sapply` function. `sapply`, used to avoid an explicit loop, returns an array containing the fields for all of the records in the file.
- iii. Once the records are read, successively the regular expressions are used to filter the records using the `apply` function.
- iv. The last statement packages the array as a dataframe object.

Below are some examples of its use.

1. To list the US insured mortality age last tables for male smokers, one could use  
`TblSearch("(?i)[^e]male.*[^\n]smoker.*last", "US")`
2. For US insured mortality age last basic tables of 1980, one could use  
`TblSearch("(?i)1980.*basic.*nearest", "US", "1")`
3. To list just the names of all US annuitant mortality tables, one could use  
`TblSearch("", "US", "2")$Name`

The second function, `Tbl`, given a vector of either table numbers or offsets returns a list of lists containing all the fields for each table requested. If the input is a vector of table numbers then `TblSearch` is used to get the offsets. The code could have been a lot shorter but for the need to have this function vectorized - this way there is just a single read of the file for all the tables combined. It does not return a dataframe like the earlier function because the mortality rates data is of varying sizes. To facilitate recursive indexing of the list of lists, the following function has proved useful. It is left uncommented as the logic is rather straightforward.

```
"RI" <-
function (z, Name)
{
  sapply(1:length(z), function(x) {
    z[[c(x, grep(paste("^", Name, "$", sep = ""), c("Name",
      "Number", "Type", "Contributor", "Source", "Volume",
      "ObsnPeriod", "ObsnUnit", "Method", "Reference",
      "Comments", "MinAge", "MaxAge", "SelPeriod", "MaxSelAge",
      "NumDec", "Rates", "HashValue", "Country", "Usage"),
      perl = TRUE))]]
  })
}
```

## RCode for Tbl

```

"Tbl" <-
function (Offset, Num = TRUE)
{
  FT <- c(1, 2, array(1, 9), array(2, 5), 3, 2, 1, 2)
  Table <- vector(length(Offset), mode = "list")
  Len <- 0
  z <- file("tables.dat", "rb")
  if (Num) {
    Offset <- TblSearch("", "", "", paste("^", as.character(Offset),
      "$", sep = "", collapse = "|"))$Offset
  }
  Offset <- sort(Offset)
  Offset <- Offset - c(0, Offset[-length(Offset)])
  for (i in 1:length(Offset)) {
    readChar(z, Offset[i] - Len)
    Len <- 0
    F <- 0
    T <- vector(20, mode = "list")
    attributes(T) <- list(names = c("Name", "Number", "Type",
      "Contributor", "Source", "Volume", "ObsnPeriod",
      "ObsnUnit", "Method", "Reference", "Comments", "MinAge",
      "MaxAge", "SelPeriod", "MaxSelAge", "NumDec", "Rates",
      "HashValue", "Country", "Usage"))
    while (F != 9999) {
      F <- readBin(z, integer(), size = 2)
      Len <- Len + 2
      if (F != 9999) {
        l <- readBin(z, integer(), size = 2)
        Len <- Len + 2 + l
        T[[F]] <- switch(FT[F], readChar(z, l), readBin(z,
          integer(), size = l), as.array(readBin(z, double(),
            n = l/8, size = 8)))
      }
    }
    Table[[i]] <- T
  }
  close(z)
  Table
}

```

Below are some queries, similar to the ones above, using the additional fields from `tables.dat`. Examples of life contingency calculations will be part of the next section.

1. To find the names of all select and ultimate US insured mortality tables, one could use
 

```
RI((y<-Tbl(TblSearch("", "US", "1")$Offset, FALSE)) [sapply(1:length(y), function(x){
  y[[x]]$Type})=="S"], "Name")
```
2. The following lists the names of insured aggregate mortality tables whose rates do not satisfy the monotonicity beyond the age 35.
 

```
x<-(y<-Tbl(sort(TblSearch("", "", "1")$Offset), FALSE)) [sapply(1:length(y), function(x){
  y[[x]]$Type})=="A"]
RI(x[as.logical(1-sapply(x, function(z) {min((z$Rates[-length(z$Rates)]<=z$Rates[-1])
  [-(1:(35-z$MinAge)]))})]), "Name")
```

## 4 Vectorization

Most actuarial quantities on the life side satisfy a linear difference equation of the first order. On compiled languages, a simple loop is the way to go and the direction in most actuarial problems happens to be backward, in some forward and in the rest one could choose either one. This is the way one computes on any spread sheet platform too - the loop becoming relative references to the preceding or succeeding cell.

When working with an interpreted vector language, explicit loops are inefficient in the presence of an algorithm which is able to *vectorize* the problem. It is not that vectorization removes the loop but rather that it makes the loop implicit i.e. it is executed internally. In the actuarial literature, see Shiu (1987), Shiu & Seah (1987) and Giles (1993), it has been noted that the usual *closed form* solution of the linear difference equation translates to elegant (concise) APL code. More than the brevity of the APL code that the closed form yields to a vectorized solution is important. In other words, as in Shiu (1987), that the closed form solution can be translated to a code which avoids explicit loops is the key. Hence this solution will lead to not only concise but an efficient solution to the problem on all vector languages where explicit loops are inefficient. Below are the details.

Life contingencies abounds with difference equations of the first order. Some are listed below.

1. The curtate future lifetime of a life aged ( $x$ ) satisfies,

$$e_x = p_x + p_x e_{x+1}$$

2. The actuarial present value of a whole life insurance with benefits payable at the end of the year of death satisfies,

$$A_x = vq_x + vp_x A_{x+1}$$

3. Fackler's formula for the reserves of a fully discrete annual whole life on  $(x)$  satisfies,

$${}_nV_x = vq_{x+n} - P_x + vp_{x+n} {}_{n+1}V_x$$

4. Hattendorff's formula for the variance of the loss random variable underlying a fully discrete annual whole life on  $(x)$  satisfies,

$$\text{Var}({}_nL|K(x) \geq n) = v^2 q_{x+n} p_{x+n} (1 - {}_{n+1}V_x)^2 + v^2 p_{x+n} \text{Var}({}_{n+1}L|K(x) \geq n+1)$$

All of the above are backward in nature as the boundary value at the right end is known. Moreover, they are each just a particular case of the equation

$$x_n = a_n + b_n x_{n+1}, \quad n = 1, 2, \dots, k \text{ with } x_{k+1} \text{ known}$$

It is easy to show that the solution is given by,

$$x_n = \frac{x_{k+1} \prod_0^k b_i + \sum_{l=n}^k a_l \prod_0^{l-1} b_i}{\prod_0^{n-1} b_i}, \quad \text{where } b_0 = 1 \text{ and } n = 1, 2, \dots, k$$

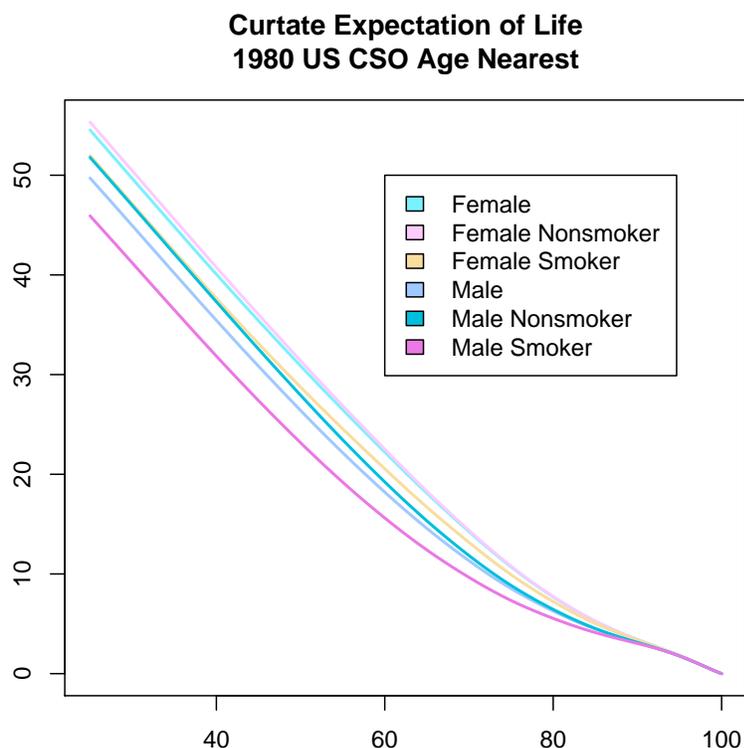
The R code for the above, which for matter of style has not been compressed into a single line, is encapsulated by the function BD.

```
"BD" <-
function (a, b, bv)
{
  s <- rev(cumprod(c(1, b)));
  (rev(cumsum(s[-1] * rev(a))) + s[1] * bv)/rev(s[-1])
}
```

As an example, below is an R code for a plot of the curve of curtate life expectation for each of the CSO 1980 basic age nearest tables. Note that the curve for female smokers is higher than that of male non-smokers.

```
y<-c("#7AF1FE", "#FCCBFF", "#F8DD9E", "#9FC9FF", "#00BFDD", "#EB78E4")
x<-Tbl(TblSearch("(?i)1980.*basic.*nearest", "US", "1")$Offset, FALSE)
matplot(25:100, sapply(x, function(z) {BD(a, a<-(1-z$Rates[-c(1:(25-z$MinAge))]), 0)}), type="l",
  lwd=2, ylab="", xlab="x", lty=1, col=y, main="Curtate Expectation of Life\n 1980 US CSO Age
  Nearest")
legend(x=70, y=50, gsub("(1980 US CSO Basic|Age nearest)", "", RI(x, "Name")), fill=y,
  horiz=FALSE)
```

## 5 Conclusion



As some of the other interpreted vector languages, R is a fantastic tool for rapid prototyping; besides, it offers a large coherent set of utilities. This combination of a prototyping language with an environment could be particularly desirable by those who compute as part of their job while not being responsible for development of software systems. In fact, R allows one to smoothly move from a prototype to a compiled code as it allows for one to access such code from within R.

For an actuarial science program, a powerful case can be made for adoption of R as the standard software across courses.

## Acknowledgement

I thank Elias Shiu and Luke Tierney for valuable discussions. All opinions and errors are mine alone. This article has been written using mainly Con<sub>T</sub>E<sub>X</sub>t (a T<sub>E</sub>X macro-package), Metapost, PERL and R. I am greatly indebted to all involved in their development.

## References

- GILES, T. L. 1993. Life Insurance Application of Recursive Formulas. *Journal of Actuarial Practice* 1, No. 2:141-151..
- SHIU, E. S. W. 1987. Discussion of "Life Insurance Transformations" by Douglas A. Eckley. *Transaction of the Society of Actuaries* XXXIX, 17-18.
- SHIU, E. S. W. and SEAH, E. 1987. Discussion of "Financial Accounting Standards No. 87: Recursion Formulas and other Related Matters" by Barnet N. Berin and Eric P. Lofgren Douglas A. Eckley. *Transaction of the Society of Actuaries* XXXIX, 37-40.

# Exhibit D

# Scenarios for Using R within a Relational Database Management System Server

Duncan Temple Lang  
Bell Labs

April 12, 2001

## Abstract

We describe an approach for performing general statistical analysis and computations directly within a relational database server rather than the more typical approach of transferring data to the client from the server and having the client perform the analyses. We outline some of the advantages of embedding the R statistical environment (language, interpreter and libraries) within the Postgres server over the usual client-processing approach. We give examples of the three classes of functions currently supported by this embedding approach.

## 1 Introduction

The reliance on databases has increased dramatically in the past few years. This is likely to increase due to the continuing escalation of the volume of data that we acquire and store. Statisticians have recognized the urgent need to communicate with these Relational Database Management Systems (RDBMS) and improved support has been added to environments such as R[2], and integration into the statistical language in the Omega language. However, support within these languages focus on access for statisticians and data analysts. It is more common that non-data analysts will be accessing the RDBMS servers and would benefit from access to statistical methodology. This is an inversion of the classical way we as statisticians have thought about statistical methodology and RDBMS. We think of embedding access to the RDBMS in the statistical software. We are suggesting embedding statistical software within the RDBMS.

In programming terms, the difference in the approach we discuss here can be succinctly illustrated in the two pseudo commands

```
f( SELECT  x FROM table )  client-side
SELECT f(x) FROM table    server-side
```

In the first case, the data is returned to the client via the SELECT query, and then the client operates on the values via the function  $f()$ . In the second approach, the server performs the per-value processing and returns just the result.

The obvious benefits from making statistical software directly accessible from within RDBMS servers include

- users can employ the familiar SQL language to invoke functions implemented in the statistical language without needing to understand that these functions are not built-in SQL functions;
- statisticians can rapidly develop and make available new methodology using their familiar;
- performance improvements accrue since we do not transfer the data to the client and process there, but perform the data reduction on the server and transfer only that reduced data.
- from regular clients, we can add new code to the statistical environment within the server
- there is increased potential for caching results that would ordinarily be computed in different client applications and thus reducing query time.

As with most ideas these days, embedding statistical software within an RDBMS is not an approach that dominates the other. There are numerous trade-offs that have to be understood before deciding on a strategy. The benefit of the approach is that “we” – accessors of data interested in statistical methodology – have more options from which to choose the most appropriate solution. The immediate benefit of embedding R within Postgres [1] is that we can quickly explore these benefits and understand the trade-offs using available software. One potential reason that we may not see performance gains is the vectorized nature of S computations. Most S functions work on multiple observations in a single invocation and are efficient by passing all of the observations to native (C or Fortran) code in one step. When we use S functions for record-wise operations in Postgres, each invocation of the S function will be given just one observation. These repeated calls may incur an overhead and lose the efficiency of the vectorized operations. More investigation is needed.

In this short paper, we will outline three different scenarios in which using statistical software embedded within the server might be useful. In section 2, we show how we can use a fixed model to predict the values for a collection of records within a database. In section 3, we show how we can use the statistical environment for fitting statistical models to large quantities of data using incremental, or record-wise, algorithms. Finally, we show how we can use “trigger functions” to process data as it is entered into the database. This allows us to discard certain observations, and correct or impute fields within the record before they are stored within a table.

These examples illustrate the three different categories of functions that we can define in an SQL extension language. These are record-wise, aggregate, and trigger functions.

While our focus is on client use of the statistical environment within the database, the embedded statistical environment can also be used directly by the RDBMS server. There is potential to obtain non-trivial performance improvements by using statistical methodologies to control evaluation of queries, including caching, prediction of execution times, etc. We hope this avenue will be explored using the embedded system to facilitate rapid prototyping and experimentation.

## 2 Prediction

In this example, we assume that the data analyst has explored a training set and decided on a particular model. They have fit that model to the available data and it is now available for prediction purposes. The model may be a parametric model, for example a linear model, or may be a non-parametric fit such as CART. This model can be used within S to make predictions for one or more new records. The user of the database now decides that she wants to predict the value for different records. She does this by issuing an SQL query from her software and is returned a column of predicted values corresponding to the different records identified in her query.

The user will specify the records and variables of interest using a regular SQL command such as

```
select predict(x1,x2) from table
```

This selects all the records in the table and passes each, one at a time, to the predict function. Each call to this function returns, in this case, a single real number. These values are then available to the caller as a column in the result set of the query. A criterion for selecting different records can also be supplied in this SQL expression using the WHERE clause. The details of the invocation of the R function remain unchanged.

How do we arrange to have this query do the appropriate computations? We start with a fitted model computed off-line in an earlier R session. Let’s call this fit  $m()$ . We store this on disk using the *save()* function in R. Now, within the Postgres server, we retrieve this object and make it available to the R *session*<sup>1</sup>

While we have stored fitted model  $m()$  on disk, we could of course have stored it directly within the database. This might be stored as a binary object (blob), using its XML representation, or any other serialization approach.

Next, we register a Postgres function named predict that will forward the request to R. We do this using the CREATE FUNCTION command in Postgres. We specify the name of the function and the types of the arguments. In this case, we specify the argument types for the two columns within the table that we will pass as predictors. These are both real-valued variables so we specify these as float\*. We should note that this can be automated from within a client application. For example, using R as a client, we can issue a query to retrieve the types of the columns in the table and then generate the CREATE FUNCTION call. If we want only a subset of the columns, then we will have to identify these and it may be more expeditious to specify the signature (argument types) of the function directly. So, as we build up the SQL command to define the function, we have the following at this point:

<sup>1</sup>Currently, there is a single session within the entire Postgres session. However, in the near future there will be multiple interpreters.

```
CREATE FUNCTION predict(float8, float8)
```

We might prefer to have the function definition indicate that one could call it with an entire tuple rather than having to explicitly enumerate each column in the table. This would make it more useful within other tables and also allow the caller to specify different columns directly.<sup>2</sup>

Next, we specify the return type of the function. Again, this is real value in our example. This adds a little more to the command to to give:

```
CREATE FUNCTION predict(float8, float8) RETURNS float8
```

The next piece of the Postgres function definition involves defining the function itself. In this case, we want to call the *predict()* function, transforming the arguments from Postgres into a data frame with a single row. We pass this data frame and the previously fitted model to predict and have it return the result. To do this, we define a new (anonymous) S function that performs these steps, giving it Postgres as the string value of the AS clause.

And finally, we specify the language in which the function is actually implemented. This identifies where Postgres should dispatch (or “send”) the call. Since this is a record-wise function, we use the procedural language `pl_R`. At this point, we have all the elements of the command to define the Postgres function that will call R to perform the prediction.

```
CREATE FUNCTION predict(float8, float8) RETURNS float8
AS 'function(x1, x2) {
    predict(model, data.frame(x1=x1,x2=x2))
}'
LANGUAGE 'pl_R'
```

Before invoking this function, we must load the *model()* object into the R session. We can do this using the load function that we have defined earlier as part of the `pl_R` Postgres extension language.

```
SELECT load('/home/duncan/model.RData');
```

In the future, we will define this function in a more generic manner so that it takes a tuple with arbitrary variables and constructs the data frame using the names and corresponding values of these variables. In other words, we will have something like

```
function(tuple) {
  predict(model, as.data.frame(tuple))
}
```

The tuple is effectively a named list and so the call to *as.data.frame()* will create a data frame with one row.

Calling *predict()* within the server has allowed us to avoid transferring data from the server to the client and to operate on the data in-place on the powerful server. Additionally, we have been able to use existing software (the prediction function) *without modification*.

## 2.1 Multiple Models and Closures

The reader might ask how we could handle prediction for two different models. Since the function we defined uses a global variable (*m()*), we would have to ensure that we had assigned the fitted model to this variable before we execute the query. This reliance on global variables is bad software design. While it would be better to pass the fitted model in the query, this is not feasible given the way SQL and Postgres work. Instead, we want each instance of this function to know about its own model. We can do this lexical scoping and closures. We first define a function in R that returns the *predict()* function.

```
predictGen <-
function(model) {
  f <- function(...) {
    predict(model, data.frame(...))
```

---

<sup>2</sup>We will work on this

```

    }
  return(f)
}

```

Now, when we call the *predictGen()* function, we get a new instance of the function it returns. Each of these has access to the *model()* object with which the *predictGen()* was called. This allows us to create different prediction functions that can work with different models.

Now, instead of supplying the definition of an R function within the AS clause of the Postgres function definition, we specify an R expression that calls *predictGen()*. We do this as

```

CREATE FUNCTION predict(float8, float8) RETURNS float8
  AS 'predictGen(load('model.Rdata'))'
  LANGUAGE 'pl_R'

```

(Notice the double quotes within the outer " pair). The AS in this Postgres command calls *predictGen()* with an R object that is loaded from a serialized version of the previously fitted model. We can create a different Postgres function, say *predict1*, in the same way but specifying a different file name in the call to *load()* from which to read the previously fitted model.

### 3 Record-wise Model Fitting: Regression

In this example, we process multiple records and compute a result that is aggregation of these. These types of *aggregate* functions can be used in statistical computations to fit models, compute statistics for groups of observations. In this example, we use

Many statistical algorithms can be readily computed record-wise. A challenge is to program these and handle others. We do not need to work on individual records but can gather blocks of observations.

Our example will compute the median of a variable. We use a statistical approach to estimate the median rather than store all the values in memory. This is based on work [?] by John Chambers, David James, Dianne Lambert and Scott Vander Wiel and uses the R package developed by David James. The package uses a class of "object" that supports methods for merging new observations with those previously processed and updating the estimate of the quantiles of the variable. The constructor or generator function is named *agentIQ()* and its first method is *merge()* and the second is *median()*<sup>3</sup>

We define the aggregator function with the following command.

```

CREATE AGGREGATE rquantile(
  basetype=float8,
  initcond1='agentIQ(100)'
  finalfunc=pl_ragg_float8_result,
  stype1=RAggregator,
  sfunc1=r_update_float8,
);

```

The important elements of this command are the first 2, and the others follow closely from this information. We firstly specify that the aggregator works on real numbers ((float8)). This establishes the type of the variable being processed.

Next, we specify how to create the object that is to be updated with each record. This is an R command which should return such an object. In this case, the function *agentIQ()* is called and the buffer size is given as 100. For each collection or group of records, this R expression will be evaluated. Therefore, for each invocation of the *rquantile* function, or for each group of records in a GROUP BY query, we will create a new instance of the updating object. Each of these will act independently of the others.

Since we this aggregator function is dealing with individual float8 values, it is natural to use the *r\_update\_float8* function as the one to do the record-wise updating. This is a function provided by the embedded R facility and is used for updating an updatable R object with one real value. We make the broad class of updatable R objects known

<sup>3</sup>This uses a slightly modified version of R package in order to simplify the example.

to R as the Postgres data type RAggregator. This is a general R object, but usually provided in the form of a list of functions defining a closure. It should have at least two methods. The first is the update function which is called for each record in the query with the value(s) in the SELECT clause. The second function is the one that is called to get the result when all the records in the group have been processed. This takes no arguments and is expected to return the aggregated result. In our case, this is the *median()* function.

One need not have these functions as the first and second elements in the list of functions returned by the initialization expression. Alternatively, one can specify them as named elements in the list with the names `update` and `result` respectively.

In our example, we modified the function *agentIQ()* so that the second element in the list of functions it returns was the final function that returned the estimate of the median. The unaltered package has the *quantile()* function as the second element of the list. We cannot use that since we have to specify which quantile we are interested in.

One approach is to append a *result()* function to the list returned by the call to *agentIQ()*. We do this in the `initcond1` attribute in the call to CREATE AGGREGATE. In our example, we would have something like

```
CREATE AGGREGATE rquantile(
  basetype=float8,
  finalfunc=pl_ragg_float8_result,
  stype1=RAggregator,
  sfunc1=r_update_float8,
  initcond1=' tmp <- agentIQ(100)
              f <- function() { quantile(.5)}
              environment(f) <- environment(tmp{\Tt{}}1)
              c(tmp, result= f)'
);
```

Note how we have to change the environment of the new function so that it can see the other functions and variables within the closure. We can of course write a function to hide the details of this merging of a new function into a closure. We might replace the four R expressions with a single expression of the form

```
addFunction(agent(100), result= function() { quantile(.5)})
```

## 4 Data Collection and Filtering

In many situations, data is gathered dynamically, processed by different applications and added to a table in a database. Observations are collected from devices connected to machines and relayed to the database. This is common in manufacturing and gathering web traffic information such as click-streams.

We consider a simple example in which we verify the values within the tuple being inserted into a table. We check that the value of a particular real-value field in the tuple falls in the appropriate range. We start with a table that has, for simplicity, three fields: `identifier`, `age` and `startDate`. The client application will issue an SQL command such as

```
INSERT INTO table VALUES ('123456', 55, '2001-03-17');
```

to put the triple of an identifier, age and starting date into the table. In this study, we limit the age of the participants to be between 30 and 39. Therefore, in this case, we want to reject the tuple.

To do this, we want to associate a function that gets called each time a tuple is entered into the table. The function might look something like the following:

```
function(tuple) {
  age <- tuple{\Tt{}}"age"
  if(age < 30 || age > 39)
    return(NULL)

  return(tuple)
}
```

This extracts the value of the `age` variable from the tuple that is being inserted and then checks whether the value is in the specified range. If it is not acceptable, we return **NULL** to signal to Postgres that it should abandon the insertion. Otherwise, we return the tuple that was handed to the function and Postgres will proceed to insert the record.

Now that we have the function, we need only arrange for it to be called. We first define it as a trigger function, and then we associate it with insertion events on the table of interest. We define the function as

```
CREATE FUNCTION checkAge() RETURNS OPAQUE
AS ' function(triggerInfo) {
    age <- triggerInfo$tuple["age"]
    if(age < 30 || age > 39)
        return(NULL)

    return(tiggerInfo$tuple)
}' LANGUAGE 'pl_R';
```

The `S` function takes a reference to the trigger information provided by Postgres. Inside in this is the actual tuple that is being inserted into the table. There are a variety of functions that allow us to get and set the names and values of the variables in the tuple, etc. In this case, we just extract the value of the `age` variable and check that this is within the range. If not, we abandon the insertion by returning **NULL**. Otherwise, we just return the tuple value that was given to us.

Finally, we register the trigger with the table. In this case, we want to be notified before the values are actually inserted so that we can veto it. Therefore, we qualify the trigger event with the **BEFORE** keyword.

```
CREATE TRIGGER foo BEFORE INSERT ON table
FOR EACH ROW EXECUTE PROCEDURE checkAge();
```

Now, issuing the insertion “queries”

```
INSERT INTO table VALUES ('123456', 55, '2001-03-17')
INSERT INTO table VALUES ('100056', 32, '2000-12-10')
```

results in the first being rejected and the second accepted. We check this by issuing the query

```
SELECT count(*) from table;
```

before and after the insertions.

Note that in this case, we are not using R’s statistical capabilities in the function. We are merely using it as a convenient, high-level scripting language. Other useful examples of triggers do use the statistical functionality. We might update aggregate statistics (means, correlation matrices, model fits, etc.) about the table. Alternatively, we might performing transformations on the tuple’s elements such as histogram equalization or coordinate registration for images, and so on. Triggers can also be defined for deletions from tables allowing us to perform the same sort of “updates” but as data is removed.

## 5 Stand-alone Functions

We have discussed using R functions in terms of operations on tables. Record, aggregate and trigger functions each operate on records associated with tables. However, we can call record-oriented Postgres functions directly and independently of a table. For example, we can invoke the `gamma` function provided in the examples shipped with the package via the query

```
SELECT gamma(4);
```

Similarly, we can define other stand-alone, or table independent, functions that can be implemented using R functions. For example, we can define functions to manage the R session and interpreter. These are regular Postgres functions that are members of the `pl_R` language and defined as in section 2.

We can provide a function for attaching and detaching R packages/libraries. This might be defined as

```
CREATE FUNCTION library(text) RETURNS int4
  AS 'function(x) {library(x); T}'
  LANGUAGE 'pl_R';
```

Similarly, we can provide functions for examining the variables in the global environment using the `objects()` function. In this case, it returns an array of strings – the names of the variables – and is a simple, direct call to objects and so needs no AS clause. The declaration can be given as

```
CREATE FUNCTION objects() RETURNS _text AS '' LANGUAGE 'pl_R';
```

This support for reflectance on the session allows privileged users to *externally* monitor and repair the R interpreter running within the server. This will hopefully allow non-intrusive diagnostic and maintenance actions without having to restart the server.

## 6 Current Status

We have developed software for embedding the statistical environment R within the Postgres RDBM server, and also the Omegahat and Java interpreters within MySQL. I currently feel that the RDBMS software on which we should focus our efforts are Postgres and Oracle. MySQL was not designed to admit such extensions. While it has been possible to add them, installing them requires modifying the MySQL code. This makes maintaining the extension complex and supporting different variants more time consuming.

This package follows our original work to modify the internals of MySQL (version 3.23.16) to support user defined functions (UDF) implemented in interpreted languages, specifically Java and Omegahat. Because MySQL was not originally designed to be extensible in this manner, and also does not support the rich set of object features that Postgres does, it is likely that we will not pursue the MySQL approach as part of our research. We encourage any interested to contact us and perhaps use the code we have developed. We will probably focus on extending the embedded language approach within both Postgres and Oracle.

## 7 Future Work

We have not mentioned any details about performance improvements that can be achieved using this embedding approach. This requires careful attention and an appropriate experimental design to account for the numerous factors that will influence the performance of the embedding and the client approaches. The obvious factors include available network bandwidth; the computational resources of the server (CPU(s), RAM, I/O speed); the average number of concurrent queries and the load on the server; the size of the tables being accessed; and so on.

A glaring omission in the current version of the package is the ability to convert R objects to Postgres arrays. This will be added shortly.

Minor enhancements to the code can be made for use with Postgres 7.1 which should realize large performance improvements. The ability to locate the S function just once per query and hence avoid the cost of per-record lookup should be quite significant in performance terms.

We may also explore storing S objects within the database itself, using binary, XML and text representations to serialize the objects. Additionally, we will explore using S objects and classes to exploit Postgres's extensible data types at the user level. (We already use this extensibility in defining the RAggregator type.) As mentioned above, we will also explore how we can pass a tuple as an argument to the record and aggregate handler functions. It is not clear that this is always possible.

We may also add functionality to R so that the R interpreter running within the Postgres server can access the tables within the server. This involves creating an interface between R and the Postgres Server Programming Interface (SPI). The tuple access in the trigger functions already uses this, and it is reasonably straightforward to add explicit support at the S language-level for the entire interface.

There are many enhancements that are needed to the statistical systems before one can deploy the embedded R within the RDBMS server approach in “production” systems. Most RDBMS servers are multi-threaded, while most statistical software is not. We are in the process of making R support multiple interpreters, and then hopefully concurrency/parallelism [3]. We need to add a security infra-structure to R so that users invoking R functions have

limited access to low-level system functions. They should not be able to access data in other concurrent R interpreters within the database. Nor should users be able to load their own C code or execute calls to the underlying operating system (e.g. using *system()*). And we must have a mechanism to identify and prohibit denial-of-service (DoS) attacks caused by consuming the servers resources (CPU cycles, disk space and access, etc.) These are active areas of research for some of us.

## References

- [1] The Postgres development team. Postgresql. <http://www.postgresql.org>, valid April 2001.
- [2] The R development team. R. <http://www.r-project.org>, valid April 2001.
- [3] Luke Tierney. Threading and GUI Issues for R. <http://www.stat.umn.edu/~luke/R/thrgui/thrgui.pdf>, March 2001.

# Exhibit E

# R Data Import/Export

---

Version 2.13.0 (2011-04-13)

R Development Core Team

---

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions, except that this permission notice may be stated in a translation approved by the R Development Core Team.

Copyright © 2000–2010 R Development Core Team

ISBN 3-900051-10-0

# Table of Contents

<b>Acknowledgements</b> .....	<b>1</b>
<b>1 Introduction</b> .....	<b>2</b>
1.1 Imports.....	2
1.1.1 Encodings.....	3
1.2 Export to text files.....	3
1.3 XML.....	5
<b>2 Spreadsheet-like data</b> .....	<b>7</b>
2.1 Variations on <code>read.table</code> .....	7
2.2 Fixed-width-format files.....	10
2.3 Data Interchange Format (DIF).....	10
2.4 Using <code>scan</code> directly .....	10
2.5 Re-shaping data .....	11
2.6 Flat contingency tables.....	12
<b>3 Importing from other statistical systems</b> ....	<b>14</b>
3.1 EpiInfo, Minitab, S-PLUS, SAS, SPSS, Stata, Systat .....	14
3.2 Octave.....	15
<b>4 Relational databases</b> .....	<b>16</b>
4.1 Why use a database?.....	16
4.2 Overview of RDBMSs.....	16
4.2.1 SQL queries.....	17
4.2.2 Data types.....	18
4.3 R interface packages .....	18
4.3.1 Packages using DBI.....	19
4.3.2 Package RODBC.....	20
<b>5 Binary files</b> .....	<b>23</b>
5.1 Binary data formats.....	23
5.2 dBase files (DBF).....	23
<b>6 Connections</b> .....	<b>24</b>
6.1 Types of connections.....	24
6.2 Output to connections .....	25
6.3 Input from connections.....	25
6.3.1 Pushback .....	26
6.4 Listing and manipulating connections .....	27
6.5 Binary connections.....	27
6.5.1 Special values .....	28

<b>7</b>	<b>Network interfaces</b> .....	<b>29</b>
7.1	Reading from sockets.....	29
7.2	Using <code>download.file</code> .....	29
7.3	DCOM interface.....	29
7.4	CORBA interface.....	29
<b>8</b>	<b>Reading Excel spreadsheets</b> .....	<b>31</b>
<b>Appendix A</b>	<b>References</b> .....	<b>32</b>
	<b>Function and variable index</b> .....	<b>33</b>
	<b>Concept index</b> .....	<b>35</b>

## 4 Relational databases

### 4.1 Why use a database?

There are limitations on the types of data that R handles well. Since all data being manipulated by R are resident in memory, and several 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) hundred megabytes in size can cause R to run out of memory, particularly on a 32-bit operating system.

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 (DBMSs) and, in particular, relational DBMSs (RDBMSs) *are* designed to do all of these things well. Their strengths are

1. To provide fast access to selected parts of large databases.
2. Powerful ways to summarize and cross-tabulate columns in databases.
3. Store data in more organized ways than the rectangular grid model of spreadsheets and R data frames.
4. Concurrent access from multiple clients running on multiple hosts while enforcing security constraints on access to the data.
5. Ability to act as a server to a wide range of clients.

The sort of statistical applications for which DBMS might be used are to extract a 10% sample of the data, to cross-tabulate data to produce a multi-dimensional contingency table, and to extract data group by group from a database for separate analysis.

Increasingly OSes are themselves making use of DBMSs for these reasons, so it is nowadays likely that one will be already installed on your (non-Windows) OS. **Akonadi** is used by KDE4 to store personal information and uses MySQL. Several Mac OS X applications, including Mail and Address Book, use SQLite.

### 4.2 Overview of RDBMSs

Traditionally there had been large (and expensive) commercial RDBMSs (**Informix**; **Oracle**; **Sybase**; **IBM's DB2**; **Microsoft SQL Server** on Windows) and academic and small-system databases (such as MySQL, PostgreSQL, Microsoft Access, . . .), the former marked out by much greater emphasis on data security features. The line is blurring, with the Open Source MySQL and PostgreSQL having more and more high-end features, and free 'express' versions being made available for the commercial DBMSs.

There are other commonly used data sources, including spreadsheets, non-relational databases and even text files (possibly compressed). Open Database Connectivity (ODBC) is a standard to use all of these data sources. It originated on Windows (see <http://www.microsoft.com/data/odbc/>) but is also implemented on Linux/Unix/Mac OS X.

All of the packages described later in this chapter provide clients to client/server databases. The database can reside on the same machine or (more often) remotely. There is an ISO standard (in fact several: SQL92 is ISO/IEC 9075, also known as ANSI X3.135-1992, and SQL99 is coming into use) for an interface language called SQL (Structured Query Language, sometimes pronounced ‘sequel’: see Bowman *et al.* 1996 and Kline and Kline 2001) which these DBMSs support to varying degrees.

### 4.2.1 SQL queries

The more comprehensive R interfaces generate SQL behind the scenes for common operations, but direct use of SQL is needed for complex operations in all. Conventionally SQL is written in upper case, but many users will find it more convenient to use lower case in the R interface functions.

A relational DBMS stores data as a database of *tables* (or *relations*) which are rather similar to R data frames, in that they are made up of *columns* or *fields* of one type (numeric, character, date, currency, . . .) and *rows* or *records* containing the observations for one entity.

SQL ‘queries’ are quite general operations on a relational database. The classical query is a SELECT statement of the type

```
SELECT State, Murder FROM USArrests WHERE Rape > 30 ORDER BY Murder
```

```
SELECT t.sch, c.meanses, t.sex, t.achieve
FROM student as t, school as c WHERE t.sch = c.id
```

```
SELECT sex, COUNT(*) FROM student GROUP BY sex
```

```
SELECT sch, AVG(sestat) FROM student GROUP BY sch LIMIT 10
```

The first of these selects two columns from the R data frame `USArrests` that has been copied across to a database table, subsets on a third column and asks the results be sorted. The second performs a database *join* on two tables `student` and `school` and returns four columns. The third and fourth queries do some cross-tabulation and return counts or averages. (The five aggregation functions are COUNT(\*), SUM, MAX, MIN and AVG, each applied to a single column.)

SELECT queries use FROM to select the table, WHERE to specify a condition for inclusion (or more than one condition separated by AND or OR), and ORDER BY to sort the result. Unlike data frames, rows in RDBMS tables are best thought of as unordered, and without an ORDER BY statement the ordering is indeterminate. You can sort (in lexicographical order) on more than one column by separating them by commas. Placing DESC after an ORDER BY puts the sort in descending order.

SELECT DISTINCT queries will only return one copy of each distinct row in the selected table.

The GROUP BY clause selects subgroups of the rows according to the criterion. If more than one column is specified (separated by commas) then multi-way cross-classifications can be summarized by one of the five aggregation functions. A HAVING clause allows the select to include or exclude groups depending on the aggregated value.

If the SELECT statement contains an ORDER BY statement that produces a unique ordering, a LIMIT clause can be added to select (by number) a contiguous block of output

rows. This can be useful to retrieve rows a block at a time. (It may not be reliable unless the ordering is unique, as the `LIMIT` clause can be used to optimize the query.)

There are queries to create a table (`CREATE TABLE`, but usually one copies a data frame to the database in these interfaces), `INSERT` or `DELETE` or `UPDATE` data. A table is destroyed by a `DROP TABLE 'query'`.

Kline and Kline (2001) discuss the details of the implementation of SQL in Microsoft SQL Server 2000, Oracle, MySQL and PostgreSQL.

### 4.2.2 Data types

Data can be stored in a database in various data types. The range of data types is DBMS-specific, but the SQL standard defines many types, including the following that are widely implemented (often not by the SQL name).

<code>float(p)</code>	Real number, with optional precision. Often called <code>real</code> or <code>double</code> or <code>double precision</code> .
<code>integer</code>	32-bit integer. Often called <code>int</code> .
<code>smallint</code>	16-bit integer
<code>character(n)</code>	fixed-length character string. Often called <code>char</code> .
<code>character varying(n)</code>	variable-length character string. Often called <code>varchar</code> . Almost always has a limit of 255 chars.
<code>boolean</code>	true or false. Sometimes called <code>bool</code> or <code>bit</code> .
<code>date</code>	calendar date
<code>time</code>	time of day
<code>timestamp</code>	date and time

There are variants on `time` and `timestamp`, with `timezone`. Other types widely implemented are `text` and `blob`, for large blocks of text and binary data, respectively.

The more comprehensive of the R interface packages hide the type conversion issues from the user.

## 4.3 R interface packages

There are several packages available on CRAN to help R communicate with DBMSs. They provide different levels of abstraction. Some provide means to copy whole data frames to and from databases. All have functions to select data within the database via SQL queries, and to retrieve the result as a whole as a data frame or in pieces (usually as groups of rows).

All except **RODBC** are tied to one DBMS, but there has been a proposal for a unified ‘front-end’ package **DBI** (<http://developer.r-project.org/db>) in conjunction with a ‘back-end’, the most developed of which is **RMySQL**. Also on CRAN are the back-ends **ROracle**, **RPostgreSQL** and **RSQLite** (which works with the bundled DBMS **SQLite**, <http://www.sqlite.org>), **RJDBC** (which uses Java and can connect to any DBMS that

has a JDBC driver) and **RpgSQL** (a specialist interface to PostgreSQL built on top of RJBDC).

The BioConductor project has updated **RdbiPgSQL** (formerly on CRAN ca 2000), a first-generation interface to PostgreSQL.

**PL/R** (<http://www.joeconway.com/plr/>) is a project to embed R into PostgreSQL.

### 4.3.1 Packages using DBI

Package **RMySQL** on CRAN provides an interface to the MySQL database system (see <http://www.mysql.com> and Dubois, 2000.). The description here applies to version 0.5-0: earlier versions had a substantially different interface. The current version requires the **DBI** package, and this description will apply with minor changes to all the other back-ends to **DBI**.

MySQL exists on Unix/Linux/Mac OS X and Windows: there is a ‘Community Edition’ released under GPL but commercial licenses are also available. MySQL was originally a ‘light and lean’ database. (It preserves the case of names where the operating file system is case-sensitive, so not on Windows.)

The call `dbDriver("MySQL")` returns a database connection manager object, and then a call to `dbConnect` opens a database connection which can subsequently be closed by a call to the generic function `dbDisconnect`. Use `dbDriver("Oracle")`, `dbDriver("PostgreSQL")` or `dbDriver("SQLite")` with those DBMSs and packages **ROracle**, **RPostgreSQL** or **RSQLite** respectively.

SQL queries can be sent by either `dbSendQuery` or `dbGetQuery`. `dbGetQuery` sends the query and retrieves the results as a data frame. `dbSendQuery` sends the query and returns an object of class inheriting from "DBIResult" which can be used to retrieve the results, and subsequently used in a call to `dbClearResult` to remove the result.

Function `fetch` is used to retrieve some or all of the rows in the query result, as a list. The function `dbHasCompleted` indicates if all the rows have been fetched, and `dbGetRowCount` returns the number of rows in the result.

These are convenient interfaces to read/write/test/delete tables in the database. `dbReadTable` and `dbWriteTable` copy to and from an R data frame, mapping the row names of the data frame to the field `row_names` in the MySQL table.

```
> library(RMySQL) # will load DBI as well
## open a connection to a MySQL database
> con <- dbConnect(dbDriver("MySQL"), dbname = "test")
## list the tables in the database
> dbListTables(con)
## load a data frame into the database, deleting any existing copy
> data(USArrests)
> dbWriteTable(con, "arrests", USArrests, overwrite = TRUE)
TRUE
> dbListTables(con)
[1] "arrests"
## get the whole table
> dbReadTable(con, "arrests")
```

	Murder	Assault	UrbanPop	Rape
Alabama	13.2	236	58	21.2
Alaska	10.0	263	48	44.5
Arizona	8.1	294	80	31.0

```

Arkansas      8.8    190    50 19.5
...
## Select from the loaded table
> dbGetQuery(con, paste("select row_names, Murder from arrests",
                        "where Rape > 30 order by Murder"))
  row_names Murder
1  Colorado   7.9
2  Arizona    8.1
3 California   9.0
4   Alaska   10.0
5 New Mexico  11.4
6  Michigan  12.1
7   Nevada   12.2
8   Florida  15.4
> dbRemoveTable(con, "arrests")
> dbDisconnect(con)

```

### 4.3.2 Package RODBC

Package **RODBC** on CRAN provides an interface to database sources supporting an ODBC interface. This is very widely available, and allows the same R code to access different database systems. **RODBC** runs on Unix/Linux, Windows and Mac OS X, and almost all database systems provide support for ODBC. We have tested Microsoft SQL Server, Access, MySQL, PostgreSQL, Oracle and IBM DB2 on Windows and MySQL, Oracle, PostgreSQL and SQLite on Linux.

ODBC is a client-server system, and we have happily connected to a DBMS running on a Unix server from a Windows client, and *vice versa*.

On Windows ODBC support is part of the OS. On Unix/Linux you will need an ODBC Driver Manager such as unixODBC (<http://www.unixODBC.org>) or iODBC (<http://www.iODBC.org>: this is pre-installed in Mac OS X) and an installed driver for your database system.

Windows provides drivers not just for DBMSs but also for Excel (‘.xls’) spreadsheets, DBase (‘.dbf’) files and even text files. (The named applications do *not* need to be installed. Which file formats are supported depends on the the versions of the drivers.) There are versions for Excel and Access 2007/2010 (go to <http://download.microsoft.com>, and search for ‘Office ODBC’, which will lead to ‘AccessDatabaseEngine.exe’), the ‘2007 Office System Driver’ (the latter has a version for 64-bit Windows, and that will also read earlier versions).

On Mac OS X the Actual Technologies ([http://www.actualtechnologies.com/product\\_access.php](http://www.actualtechnologies.com/product_access.php)) drivers provide ODBC interfaces to Access databases (including Access 2007/2010) and to Excel spreadsheets (not including Excel 2007/2010).

Many simultaneous connections are possible. A connection is opened by a call to `odbcConnect` or `odbcDriverConnect` (which on the Windows GUI allows a database to be selected via dialog boxes) which returns a handle used for subsequent access to the database. Printing a connection will provide some details of the ODBC connection, and calling `odbcGetInfo` will give details on the client and server.

A connection is closed by a call to `close` or `odbcClose`, and also (with a warning) when not R object refers to it and at the end of an R session.

Details of the tables on a connection can be found using `sqlTables`.

Function `sqlSave` copies an R data frame to a table in the database, and `sqlFetch` copies a table in the database to an R data frame.

An SQL query can be sent to the database by a call to `sqlQuery`. This returns the result in an R data frame. (`sqlCopy` sends a query to the database and saves the result as a table in the database.) A finer level of control is attained by first calling `odbcQuery` and then `sqlGetResults` to fetch the results. The latter can be used within a loop to retrieve a limited number of rows at a time, as can function `sqlFetchMore`.

Here is an example using PostgreSQL, for which the ODBC driver maps column and data frame names to lower case. We use a database `testdb` we created earlier, and had the DSN (data source name) set up in `~/odbc.ini` under `unixODBC`. Exactly the same code worked using `MyODBC` to access a MySQL database under Linux or Windows (where MySQL also maps names to lowercase). Under Windows, DSNs are set up in the ODBC applet in the Control Panel ('Data Sources (ODBC)' in the 'Administrative Tools' section).

```
> library(RODBC)
## tell it to map names to l/case
> channel <- odbcConnect("testdb", uid="ripley", case="tolower")
## load a data frame into the database
> data(USArrests)
> sqlSave(channel, USArrests, rownames = "state", addPK = TRUE)
> rm(USArrests)
## list the tables in the database
> sqlTables(channel)
  TABLE_QUALIFIER TABLE_OWNER TABLE_NAME TABLE_TYPE REMARKS
1
  usarrests      TABLE
## list it
> sqlFetch(channel, "USArrests", rownames = "state")
      murder assault urbanpop rape
Alabama      13.2    236      58 21.2
Alaska       10.0    263      48 44.5
...
## an SQL query, originally on one line
> sqlQuery(channel, "select state, murder from USArrests
  where rape > 30 order by murder")
  state murder
1 Colorado   7.9
2 Arizona    8.1
3 California 9.0
4 Alaska    10.0
5 New Mexico 11.4
6 Michigan  12.1
7 Nevada    12.2
8 Florida   15.4
## remove the table
> sqlDrop(channel, "USArrests")
## close the connection
> odbcClose(channel)
```

As a simple example of using ODBC under Windows with a Excel spreadsheet, we can read from a spreadsheet by

```
> library(RODBC)
> channel <- odbcConnectExcel("bdr.xls")
## list the spreadsheets
> sqlTables(channel)
```

```
TABLE_CAT TABLE_SCHEM      TABLE_NAME  TABLE_TYPE  REMARKS
1 C:\\bdr      NA           Sheet1$     SYSTEM TABLE  NA
2 C:\\bdr      NA           Sheet2$     SYSTEM TABLE  NA
3 C:\\bdr      NA           Sheet3$     SYSTEM TABLE  NA
4 C:\\bdr      NA Sheet1$Print_Area      TABLE         NA
## retrieve the contents of sheet 1, by either of
> sh1 <- sqlFetch(channel, "Sheet1")
> sh1 <- sqlQuery(channel, "select * from [Sheet1$]")
```

Notice that the specification of the table is different from the name returned by `sqlTables`: `sqlFetch` is able to map the differences.

# Exhibit F

# Improving the analysis, storage and sharing of neuroimaging data using relational databases and distributed computing

Uri Hasson,<sup>a,b,\*</sup> Jeremy I. Skipper,<sup>a,b</sup> Michael J. Wilde,<sup>c,d</sup>  
Howard C. Nusbaum,<sup>b,e,f</sup> and Steven L. Small<sup>a,b,f</sup>

<sup>a</sup>Department of Neurology, The University of Chicago, Chicago, IL, USA

<sup>b</sup>Department of Psychology, The University of Chicago, Chicago, IL, USA

<sup>c</sup>The Computation Institute, The University of Chicago, Chicago, IL, USA

<sup>d</sup>Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439, USA

<sup>e</sup>Centre for Cognitive and Social Neuroscience, The University of Chicago, Chicago, IL, USA

<sup>f</sup>The Brain Research Imaging Centre, The University of Chicago, Chicago, IL, USA

Received 17 November 2006; revised 30 July 2007; accepted 6 September 2007

Available online 21 September 2007

**The increasingly complex research questions addressed by neuroimaging research impose substantial demands on computational infrastructures. These infrastructures need to support management of massive amounts of data in a way that affords rapid and precise data analysis, to allow collaborative research, and to achieve these aims securely and with minimum management overhead. Here we present an approach that overcomes many current limitations in data analysis and data sharing. This approach is based on open source database management systems that support complex data queries as an integral part of data analysis, flexible data sharing, and parallel and distributed data processing using cluster computing and Grid computing resources. We assess the strengths of these approaches as compared to current frameworks based on storage of binary or text files. We then describe in detail the implementation of such a system and provide a concrete description of how it was used to enable a complex analysis of fMRI time series data.**

© 2007 Elsevier Inc. All rights reserved.

---

## Introduction

The development of non-invasive neuroimaging methods, such as positron emission tomography (PET), and functional magnetic resonance imaging (fMRI), has produced an explosion of new findings in human neuroscience. Scientific advancement in this domain has been the direct result of developments both in

hardware technology for data acquisition and algorithms for data processing and image analysis. As these analytical approaches have improved in sensitivity and power, they have made it possible to address increasingly complex scientific questions. Yet, while the scientific questions and analysis methods have become more sophisticated, the computational infrastructures to support this work have generally not kept pace. In this article, we discuss a novel computational approach to support analysis of functional imaging data. The importance of this approach is that it allows neuroscientists to address more complex questions while concomitantly speeding up the rate at which these questions can be evaluated.

Early neuroimaging research was based on grouping trials of the same sort into a single presentation sequence in so-called “block designs”. While these designs enabled researchers to address certain a priori questions, they left little room for a posteriori data analysis. More recently, “event-related designs” (both slow and fast variants) have not only enabled researchers to evaluate a priori research questions but, importantly, also enabled a variety of interesting a posteriori analyses that have been of tremendous value. For example, some researchers have partitioned the stimuli according to post hoc classifications after data have been collected, as in a study by Wagner et al. (1998), which analyzed stimuli as a function of whether they were subsequently remembered or forgotten. The use of event-related designs has also opened the way to new statistical analysis methods for estimation of event-linked hemodynamic responses, and for assessing the correlation between neural activity and finer features of stimuli properties.

In light of these advancements, it is noticeable that there has been substantially less progress in the development of computational infrastructures supporting the storage, analysis, and sharing of fMRI data. Although there are significant efforts underway to

---

\* Corresponding author. Human Neuroscience Laboratory, Biological Sciences Division, University of Chicago Hospital Q300, 5841 S. Maryland Avenue MC-2030, Chicago, IL 60637, USA. Fax: +1 773 834 7610.

E-mail address: [uhasson@uchicago.edu](mailto:uhasson@uchicago.edu) (U. Hasson).

Available online on ScienceDirect ([www.sciencedirect.com](http://www.sciencedirect.com)).

represent and store imaging data for large multi-center studies (Van Horn et al., 2001), the infrastructures at individual research centers are often not optimally designed to support everyday imaging research tasks. Most importantly, the performance of increasingly complex analyses, such as evaluation of functional connectivity between brain regions, requires certain computational tasks that can be cumbersome and even prohibitively difficult using traditional data representation approaches (i.e., hierarchical file systems and matrix representation of images). Such complex analyses require, for example, repeated averaging of subsections of time series (TS) data and correlating TS data, but currently employed frameworks for data storage are ill equipped for this task. Furthermore, as the complexity of analyses increases, current approaches to data representation generate prohibitively large amounts of intermediate data (e.g., “mask” files) in addition to the final results. This in itself causes serious management overhead. The immediate result of these weaknesses is that the computational infrastructure becomes a bottleneck in the progress of research: it results in slower data analysis, reduces the number of questions that can be asked of the data, and makes it difficult to enable concurrent access to the data (for local and remote users) as is often needed for complex analyses and collaborative research. Thus, the current computational demands for imaging research call for a different approach to storage and analysis of fMRI data. The basic requirements of such systems are that they store data efficiently, enable rapid selection of data, and make data easily accessible for both local and remote users.

In what follows, we present a unified framework for the analysis, storage and sharing of neuroimaging data that addresses these needs, using an approach based on the general data representation and manipulation abilities of database management systems (DBMSs). While this framework is technical in nature, its forte is in extending the researcher’s ability to ask more questions about neuroimaging data and obtain rapid responses to these questions while employing advanced statistical tools. These advantages increase the efficiency of a scientific inquiry process that is often based on being able to ask increasingly refined questions about data.

A major advantage of the database-centric framework we present here is that it not only uses DBMSs for storing and sharing of data, but also takes advantage of DBMS capabilities by making the database an integral part of the fMRI data analysis workflow. We review the advantages that this approach offers over the traditional methods of storing and analyzing data using flat files (i.e., binary or text files), and show how these directly bear on the scientific routine and daily research in brain imaging. We demonstrate the scalability of these methods when coupled with modern distributed cluster computing (Pfister, 1998) and Grid computing technologies (Foster, 2005), in which numerous computers (computing nodes) perform tasks in parallel, and discuss issues such as efficient data storage, data sharing, data transparency, and advanced data analysis. Finally, we detail our implementation of such a system.

Our aim is to introduce such systems to researchers who have not considered this approach so that they can become acquainted with both the strengths and limitations of database-oriented analysis of brain images. We therefore first describe our general approach rather than the specific details of our implementation (in section, Relational databases and their application to imaging). We then present the description of the system’s actual implementation

(in section, System description). The system is based on open source software tools (widely available and supported by large developer communities) and a client–server approach; the data are stored using a database server, and analyzed by remote client computers, which request data over the network and analyze the data using a powerful statistical programming language (R Development Core Team, 2005; <http://www.R-project.org>). We then provide concrete details of one example analysis to communicate more practical information (in section, Detailed example: reverse correlation analysis). Specifically, we explain how this system was employed to conduct an analysis that exemplifies beneficial aspects of using DBMS in conjunction with distributed computing to conduct fMRI data analysis. This analysis is a “reverse correlation” of fluctuations in hemodynamic responses with specific stimulus properties of naturalistic stimuli. We trust that these descriptions on both abstract and concrete levels will allow researchers to consider more diverse and creative analysis methods and efficient ways for sharing and storing data.

### Relational databases and their application to imaging

As scientists wrestle with the exponential growth of their datasets, the power and utility of the relational database is being applied with increasing breadth and frequency across a range of scientific disciplines (Szalay and Gray, 2006). The benefits in terms of indexability, leveraging of metadata, and scalability of database approaches over file-based approaches are becoming clear in a growing number of disciplines (Gray et al., 2005). This trend can be seen clearly in digital astronomy, where the Sloan Digital Sky Survey (<http://www.sdss.org/>) is making an increasing use of DBMS technology to describe millions of celestial objects, and to enable searches across that data (Nieto-Santesteban et al., 2005). In this effort, improved data organization and relational representation enables database queries, performed in a distributed manner on Grid resources, to run an order of magnitude faster than a file-based implementation of the same algorithm operating over file-based catalogs.

In bioinformatics, the warehousing of file-based data from both curated public data sources and laboratory experiments into integrated relational databases affords new methods for search and analysis. Here, the Genomics Unified Schema (<http://www.gusdb.org>; cf., Davidson et al., 2001) provides a fabric for creating integrated relational databases for functional genomics data analysis from public data sources and from laboratory experiments in sequence analysis and proteomics (Stoeckert, 2005).

Researchers using imaging data are already facing similar challenges. fMRI analyses typically use and generate a vast number of data files. For example, individual participant data might include structural images optimized for different tissue parameters (e.g., T1, T2, FLAIR), diffusion-weighted images (isotropic and anisotropic), perfusion images, angiograms, surface representations of volumes, regions of interest, numerous TS (e.g., unregistered, registered, despiked, error terms), various masks, as well as numerous statistical maps. Group-level statistical maps might reflect the results of various types of statistical analyses performed on the individual level data (e.g., analysis of variance (ANOVA), principal components analysis (PCA), *t*-tests, etc.). Together, the number of flat files generated (i.e., linear unstructured data stored in files and organized in directories) can become quite large and the entire set is typically complex, difficult

to manage, and enormous in size. This is particularly so when data are kept in the form of text files for purposes of certain advanced analyses. DBMS offers many advantages over flat files in terms of storage, sharing and analysis, and we discuss some of these in what follows. Certainly flat file systems allow more rapid sequential access to data, which under the right circumstances, can result in faster processing. Yet, this advantage is less important when the data in the database are analyzed in parallel utilizing high-performance distributed computing systems.

In DBMSs, data are not stored in separate user-accessible files but are encoded in a tabular internal representation that reflects relations among data elements or tables of such elements (how or where this information is stored is irrelevant to users, and so we will not address this further). All a user needs to know in order to access the data is the name of the table storing the data and what data attributes it holds. For example, a user can request to see all the information in the *subject04* table by issuing a command (equivalent to): *show all information in table subject04*. Or, if more specific information is needed: *show all information in table subject04 where the condition is 'tone-presented'*. DBMSs are therefore indispensable for querying (i.e., asking subset and relational questions of) large amounts of data, and in the System description section we demonstrate how such capabilities can be utilized for rapid development and execution of sophisticated fMRI analyses. A number of research projects have utilized databases for archiving and making available large numbers of imaging datasets (Kotter, 2001; Van Horn et al., 2001), or the results of statistical analyses (Fox and Lancaster, 2002). Such large-scale projects, however, use DBMS to manage large amounts of file data, rather than to maintain data in a form that facilitates use in outside analysis routines. They are not aimed at affecting the daily practices of researchers working on fMRI projects in those stages of the work where data are still being analyzed (or in some cases, mined) for certain patterns. Rather, they are intended for archiving, reanalysis and meta-analysis.

For the individual researcher or a research laboratory, storing data in a database implies that given proper permissions, the data could be accessed from any remote computer (whether on the local network, or over the Internet) obviating the need to save multiple copies of data at different locations. As a result, sharing data with remote collaborators is greatly simplified, because servers can accept requests for data (queries) over computer networks. For example, two research groups can analyze the same dataset using different methods of analysis (e.g., ICA vs. contrast analysis). DBMSs also allow for data filtering on the server side, thus eliminating unnecessary network traffic. In practice, an analysis script written at one location can be sent to remote collaborators and executed from their computers without any modification whatsoever, since the remote center will access the original data, and the output of the analysis would be identical across sites independent of the complexity of the analysis or its subtleties (see Appendix for example). Furthermore, databases offer a single point-of-update: updating data on the server will immediately affect all analyses conducted on those data without the need to send newer versions of the data to other individuals involved in its analysis. Given proper coordination (updates should not occur during data analysis proper), this feature assures that all relevant parties access the exact same dataset.

Because database systems allow simultaneous access to data from multiple sources, they lend themselves to distributed computing of various types. One distributed approach involves

cluster-computing frameworks in which multiple computers (computing nodes) work in parallel to distribute the processing of a single computing job (Pfister, 1998). Another approach, termed Grid computing (Buyya et al., 2005; Foster, 2005; Foster et al., 2001), is based on more loosely associated computing groups with intelligent 'middleware' software that makes those computers appear as a single computing resource from the user's perspective. In both types of solutions, dozens or even hundreds of computers perform analysis in parallel, simultaneously accessing the same dataset (the approach described here was implemented on a computing cluster that supports Grid computing; functionality that necessitates Grid computing is highlighted in the text).

While offering the possibility of storing data at a single location, if needed, DBMSs offer integral replication features that can speed up analyses and serve as a backup mechanism. For instance, data stored on a database in a neuroimaging laboratory can be replicated to a "mirror" database (technically known as a 'slave') at a different laboratory, allowing a remote collaborator to work on a local copy of the data if needed. This scenario is particularly useful if the dataset is very large. A large raw TS dataset can consist of dozens of gigabytes that would otherwise have to be transferred over the network during each analysis. In another scenario, the slave database might be set up on the same network as a computing cluster. In this configuration, during data analysis the cluster nodes access the data on the slave database, which is located on the same local area network as the cluster and is accessible via fast (e.g., fiber or gigabit) connections (see Fig. 1). This configuration offers more efficient data access than connecting to the original database over relatively slower wide-area network connections (e.g., Internet connections). Replication can also be used to reduce the workload on a server when multiple machines need to access the database in parallel, such as when multiple nodes are processing data simultaneously. For example, 20 nodes can be configured to query the master database, and 20 others can be configured to query the slave thus offering the required scalability for parallel environments. (More sophisticated implementations, such as 'rolling out' partial copies of a database to database engines running on the computing nodes are also possible.) Finally, slave databases serve as immediately accessible backup systems if the main system becomes inaccessible.

Existing fMRI analysis tools could potentially interface with DBMS. Current data analysis systems (e.g., AFNI, SPM, BrainVoyager, FSL) are integrated packages that use flat files to save data throughout the analysis flow and allow users to invoke statistical procedures using integrated commands or extensions. Using a database as a storage 'backend' in these systems would allow users to access data via database queries (rather than from a file) thus benefiting from DBMS features described above, while still retaining a familiar working environment. In addition, many software systems and programming language (e.g., Matlab, Excel, Perl, Python, C) can currently interface with relational databases, which allows for parallelized data processing by users others than those who had collected the data.

Effective and easy documentation of data structures is a natural byproduct of data representation in DBMS. Relational databases can easily be used to serve metadata such as the names of the tables in the database, the columns (attributes) that exist in each table, and the type of data stored in each column. This feature makes it easy to document the structure of the database and facilitates more effective sharing of information with others. We now turn to

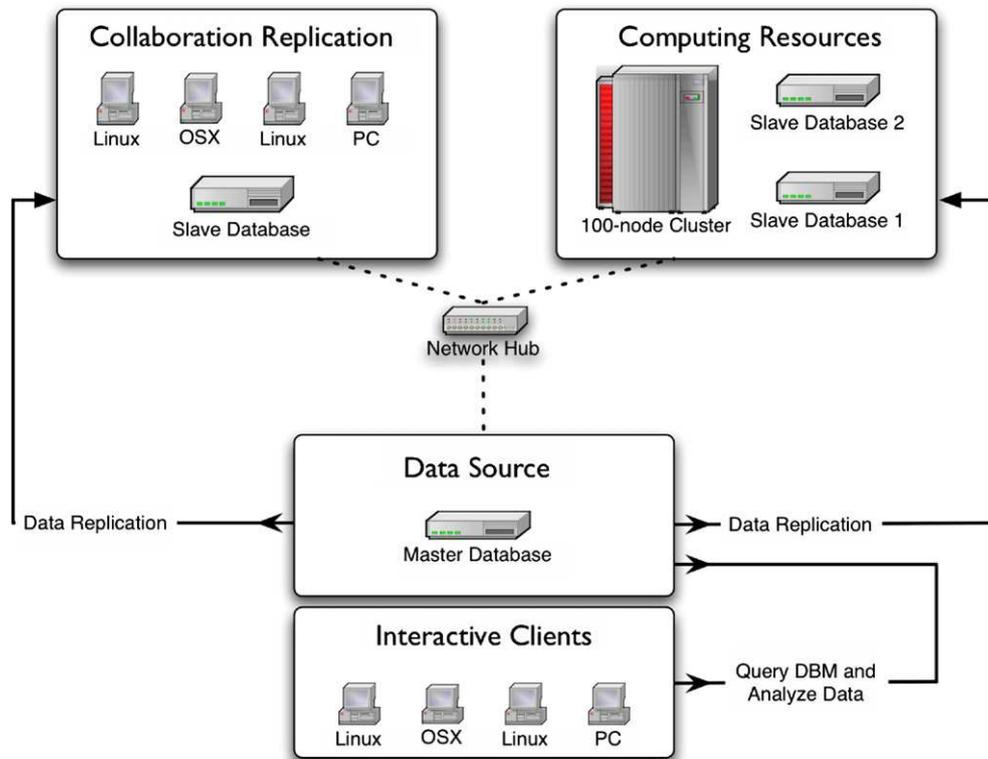


Fig. 1. Sharing and analyzing data using databases. fMRI data collected at one center (the Data Source) are stored on a Master database, and are replicated to a collaborator, as well as to a 100-node computing cluster. Collaborators can either analyze the data locally, or query data from the master database. The computing cluster holds two copies of the data using two separate DBMS servers, to serve 100 clients simultaneously.

describe the specific details of the neuroimaging data analysis system we have implemented.

## System description

### General

The system we have implemented is based on an architecture similar to that in the framework described above, in which distributed clients pull data from a central server, and work independently and simultaneously to conduct a voxel-based analysis (volume domain), a node or vertex-based analysis (surface-mapping domain; e.g., Argall et al., 2005), or a region-based analysis. In what follows we refer to voxels as a default, unless specifically referring to analyses conducted in the surface domain. The server maintains a relational database that stores the data that are to be analyzed as well as information about that data, e.g., the assignment of nodes to anatomical regions of interest (henceforth ROIs). The clients that conduct the data analysis are compatible with all major operating systems (e.g., Microsoft Windows, UNIX variants or Apple Mac OSX).

### Server

#### Data representation

In our implementation, each experiment is assigned a single database, and each database can contain a varying number of data tables. The guiding principle in designing such databases is to separate the fMRI data tables that store functional data for volume

voxels or surface vertices (e.g., BOLD data) from the tables that hold descriptive information about these voxels or vertices.

The fMRI data for each individual participant are stored in a table (or tables) that holds all data for that participant, i.e., for all voxels (in the volume domain) or nodes/vertices (in the surface domain), for all conditions.<sup>1</sup> If the data are signal estimates from a statistical analysis, such tables will have  $[N(\text{voxels}) * M(\text{conditions})]$  cells. If the data are the raw TS, the table will have  $[N(\text{voxels}) * M(\text{time points})]$  cells. For example, in an experiment with two conditions, where each hemisphere is represented as a flat surface map consisting of 196,000 vertices, data would be stored in a table with 196,000 rows, and two columns.

Theoretical descriptions (classifications) of the data that are used for filtering and selection purposes during analysis are stored in different tables in the database (see Fig. 2). These tables are used to classify voxels or surface nodes according to criteria that are of theoretical interest. For example, one such table could associate each voxel with an anatomical brain region. Such a table would contain two columns: one for the voxel number and one for the brain region descriptor (label or number). In this case, the classification can record as many values as needed in the researcher's anatomical parcellation system. Tables can also record whether a voxel is part of

<sup>1</sup> We use the term "fMRI data" to refer to two types of data. One is the actual TS data, i.e., the sequences of signals from a single voxel that are measured over the entire course of an experimental run. These data are typically mean normalized and analyzed by regression models. The second type of data is the signal estimates that are the result of statistical analyses (e.g., beta values estimated from regression or deconvolution analyses).

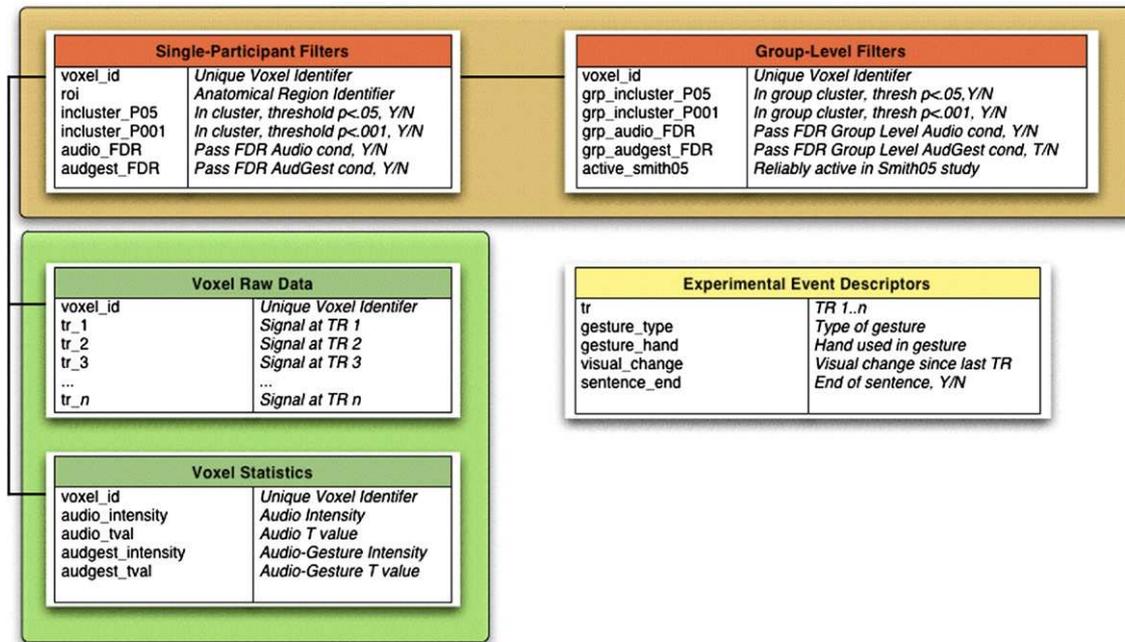


Fig. 2. Example of database scheme for storing data from an fMRI experiment. Each titled table reflects a table in the database and the information it maintains. Separate tables store the time series data and signal estimates (green). The database returns the data of voxels satisfying a certain criteria. If no criteria are specified, the data for all voxels are returned. Criteria are specified as constraints based on the filter tables (orange). Some filters are linked to individual participants (single-participant filters) whereas others are linked to the entire group of participants in the experiment (group-level filters).

a region that has certain functional properties, e.g., whether it is implicated in emotional processing as determined by an independent “localizer” task, whether its intensity passes a certain reliability criterion, whether it was found active in a certain previous study, or any other classification that is of interest to the researcher (Fig. 2).

Note that some of these filters may be linked to specific participants whereas others are not. For example, due to differences in brain structure, assignment of voxels to their anatomical regions will often be performed on an individual basis so that the relation between voxels and anatomical labels would be unique for each participant (e.g., as established via automatic parcellation: Desikan et al., 2006; Fischl et al., 1999). By contrast, classifying voxels according to whether or not they were active in a previous experiment on a group level would be represented in one table that would be applicable to all participants in the study. Finally, some classifications, such as whether a voxel demonstrated reliable intensity in a given condition, could be described on the group or individual participant level. This decision depends on whether a researcher wants to select voxels active at the group level, or those active for each participant on an individual basis (even though these are likely different voxels). In the latter case it would be necessary to identify separately for each participant which voxels were active in each experimental condition.

Once the descriptor tables have been constructed, researchers can rapidly select data according to highly specific criteria that implement one or more constraints in any logical combination. For the database in Fig. 2, it is trivial to select voxels that meet criteria such as being in the left inferior frontal gyrus, having a  $t$ -value that is greater than a certain criteria in one or more experimental conditions, or having been classified as active in a prior study. Because relational databases are designed to resolve such complex queries, it is straightforward to combine any such criteria in a query.

Consider the following query that can be constructed using a single statement to extract voxel data for a focused analysis: for each participant, extract data of voxels in the left superior temporal gyrus that are part of an active cluster at group level in the audio condition, or had a reliable  $t$ -value in that condition at the individual-participant level. This sort of query may be particularly useful when trying to establish regularities at the group level while at the same time accounting for inter-individual differences that exist in the location of activation peaks (cf. Patterson et al., 2002).

#### Server implementation

The database server software that we use is the MySQL database engine, which is freely available on the Web (<http://www.mysql.com/>), and can be installed on UNIX variants, Apple’s Mac OSX platform or Microsoft Windows. This database system has extensive documentation, use publications, and graphical interface management tools that allow it to be rapidly mastered by non-specialists. Tables can be created via graphical interfaces or command line tools, and loaded from text files. The database supports the Structured Query Language (SQL; Eisenberg et al., 2004) that is used to specify what information is to be pulled from the database. Access to the database is typically achieved via Internet protocols, so that remote data can be accessed given proper security permissions, but for development purposes, a command line mode is also available. In our work, each experiment is assigned a unique database — a collection of data tables that contain both functional data and theoretical classifications of those data as described above. The database can also function as a job dispatch manager and manage the parallelization of jobs to the computing nodes (see Appendix). This makes it possible to run one script repeatedly, while assuring that each instance of the script is initialized with different runtime operation parameters.

### Database security and controlled collaboration

Security policies on DBMS control what operations each user can perform on the data. Because the database is accessible over the network, a user's account consists of both a user name (to which a password is assigned) and a collection of hosts (i.e., terminals) from which that user can access the database. This combination assures that certain users will be able to access the database from any host, given that a password is provided, whereas others will be able to access it only from certain hosts.

Different users or groups of users can be given different rights to the data, and this is the typical approach for an fMRI study. The researcher who collected the data will likely receive all permissions to the database and remote colleagues will likely be granted more limited privileges. For example, such users should not be able to delete tables from the database, or to change their structure.

Because databases are designed with data sharing as a design principle, DBMS offers a powerful and flexible permission scheme. In MySQL, the privileges granted to an account can apply to an entire database, specific tables in the database, or even specific columns in a table. Certain users could view data in all tables in the database, whereas others could be limited to a few tables. The most basic procedures for which security would be implemented include rights to select (i.e., access) data, update data, or delete data. In many research laboratories, such security is mandated to protect the identity of subjects or patients.

Databases also offer flexible mechanisms for separating between data that are to be shared and those that are not. For various reasons, researchers are very careful with the portions of the data they share with others (cf. Ascoli, 2006), and managing the sharing of neuroimaging data is a nontrivial problem (e.g., Smith et al., 2004). To illustrate, a researcher might want to store the data of 50 participants in a database table for purposes of his or her own analyses but share only those data belonging to the subset of participants (e.g., 20 participants) whose data have been published. In a database, this is easily enabled by creating a "virtual table" (technically called a "view") that is in itself a result of a query, but that appears as a table when querying the database. In this case, the view named "limited.20ss.table" would be the result of a query selecting all data belonging to the relevant 20 participants. Other users will interact with this view as if it were a table and analyze it according to their interest (e.g., 'select all data from limited.20ss.table where condition1.tvalue>4'). Views make it possible to share data without needing to make additional custom-tailored copies of the data to suit different types of sharing. Also, when data in the primary tables are updated, these changes are immediately seen in the views (see, Gray et al. (2005), for advantages of views in the context of scientific research).

### Standards, conventions, and local practices

Given that the type of system described here is aimed at individual researchers or research laboratories, local practices will ultimately determine the structure of databases and table-naming conventions, and the nature of the metadata maintained. Though adopting a common standard aids in data sharing, in a system of the order we are describing, sharing is carried out on a peer-to-peer level (i.e., by having research centers establish direct contact), rather than via a central data warehouse that holds numerous datasets.

The development of general representation schemes that can accommodate different types of fMRI analyses and their associated data types is a matter of ongoing research (e.g., *OntoNeuroBase*,

Temal et al., 2006). Intensive work has also been conducted by the BIRN project (<http://www.nbirn.net>) to develop a logical model for documenting results of statistical analyses using XML (Keator et al., 2006). This model provides a framework for storing metadata about functional scans, functional data, and various annotations. However, it is a non-trivial task to establish a domain ontology for neuroimaging that would be readily adopted by a large number of research laboratories and aid data interoperability. On the theoretical level, one would need to establish a set of data types and characterize how these types relate to each other. Even then, it is unclear whether in practice such a general scheme would be adopted by researchers; e.g., different research centers would need to agree on a common nomenclature for naming cortical regions, possibly within a larger context of a hierarchy of brain structures (e.g., *NeuroNames*, Bowden and Martin, 1995). In the absence of such agreements, any such implementation would need to incorporate flexibility, such as accommodating multiple anatomical labelings for the same data (cf. Keator et al., 2006, for such an implementation).

In reality, the description of the data in many centers is likely to be quite idiosyncratic and even project-specific. What is important is that the database structure be accurately described, and that this description be publicly available. Once the analysis is completed and the data submitted to a central repository (e.g., fMRIDC), standard metadata conventions could be applied to the data (see, e.g., Gardner et al. (2003), for standards in central and peer-to-peer repositories).

Rather than developing a general storage scheme, during our 2.5-year experience with DBMS-driven analysis of fMRI data we instead opted to construct database schemes for different usage cases. Some schemes, as the one described in the Data representation section and Fig. 2, are quite detailed. Other schemes, supporting relatively simple analyses, contain only two tables. For example, a database set up to support analysis of a block-design experiment with three conditions (analyzed in the surface domain) would have the following fields in each table, where each field corresponds to a column in the table:

*table 1* (individual participant data): hemisphere, participant\_id [1 ... n], node\_id [1 ... 196,000], cond1\_beta [signal estimate in condition1], cond2\_beta, cond3\_beta.

*table 2* (group level descriptor): node\_id, roi\_id [anatomical region in common space], reliable\_cond1 [reliable by FDR on group level, y/n], reliable\_cond2, reliable\_cond3.

A conceptually similar study using an event-related design would have a similar table structure, except that instead of one signal estimate per each condition, the table would store the data for the estimated impulse response function (IRF) in each condition; e.g., if the IRF is estimated by 7 data points, these would be stored as cond1\_tr1beta ... cond1\_tr7beta, and so on.

If the study were extended to include two groups of participants presented with the same stimuli under different task instructions (that is, in two separate experiments), a between-participant factor (task) would be coded in an additional column in tables 1 and 2, as follows:

*table 1* (individual participant data): task, hemisphere, participant\_id, node\_id, cond1\_beta, cond2\_beta, cond3\_beta.

*table 2* (group level descriptor): task, node\_id, roi\_id, reliable\_cond1, reliable\_cond2, reliable\_cond3.

This last example illustrates how data from multiple experiments can be stored in the same table or database when such a scheme is useful for answering the theoretical question at hand. Schemes for TS analyses can also be developed, and we detail a few in the section, Detailed example: reverse correlation analysis.

Data from separate databases can be cross-referenced or joined in a single query, if those separate databases reside on the same server. This makes it possible to extract data from one study on the basis of results derived in another study. To illustrate, signal estimates could be selected only for voxels that were reliable in a certain condition in a prior study (certain commercial DBMS, e.g., MS SQL Server, also enable queries that access databases residing on different servers). This also makes it possible to create on the fly (via SQL queries) newly ‘joined’ tables from data collected in two different experiments.

The example cases we have discussed above were rapidly implemented by individuals at the graduate- and undergraduate-student level, with minimal oversight by more experienced users. These use cases show that while each study may dictate its own table organization, some general principles are emerging, such as the separation of data themselves from the descriptors of the data, which allows filtering of data from one experiment on the basis of constraints from another. Implementing similar systems in research centers would likely involve a similar process, in which experience with the system will lead to commonalities in schema design and the emergence of ‘prototypical’ schemes.

#### Data storage requirements

The data storage requirements associated with storing fMRI data in DBMS depend on a number of factors, including the number of participants, and the types of data being stored (statistical estimates such as beta coefficients, and/or entire TSs). Here we report the storage requirements for two types of example datasets when stored in a database vs. when stored in imaging file formats. The first dataset consists of signal estimates in three experimental conditions for each voxel in the volume domain (73,000 voxels per each participant). The second consists of TS data (1620 acquisitions) for each vertex in the surface domain (196,000 surface vertices per hemisphere, per participant, making for more than  $3 \times 10^8$  data points per participant).

The first dataset required ~37 MB when stored in the database (the database included indexes on two columns for faster data selection, which slightly increase its size). On the traditional hierarchical file system, it required ~3.5 MB when stored in a compressed binary format (BRIK.gz), or ~10 MB when stored in a non-compressed binary format (BRIK). In both the database and the BRIK files, the data were stored as a floating-point numeric type with precision of five decimal places. The command line utilities we routinely use are part of the AFNI suite and can perform voxel-based analysis on compressed BRIK.gz files thus benefiting from the smaller storage requirements.

The second dataset contained surface vertex data and consisted of several large TS files, one per each participant’s hemisphere, each stored as a separate table. Each file required ~3600 MB when stored in its typical form, which is as a text file (in AFNI, surface-based analyses take text files as input rather than binary files). Each corresponding database table was ~1250 MB in size (with an index on one of the columns) when stored in the database.

When considering storage requirements, it is important to note the following: first, databases offer compression options, and in

MySQL, such compression achieves between 40 and 70% reduction in data size, but entails making a table read only. The data sizes we report above are for uncompressed data. Second, storing data in compressed formats can be associated with increased processing time during data access because of the requisite decompression and recompression. Working with compressed files (e.g., BRIK.gz) via a graphical interface (e.g., the AFNI interface) can also be associated with reduced responsiveness of the interface (see <http://afni.nimh.nih.gov/pub/dist/src/README.compression>). Thus, implementing compression in either file-based or database environments should be carefully considered depending on the particular demands of each project. For instance, projects whose analysis has ended are good candidates for compression.

#### Interfaces with imaging workflow

The workflow of a typical imaging analysis consists of a large number of processing stages, often beginning from reconstructing data from *k*-space files, and culminating in thresholding. Our work to date has mainly utilized DBMS capabilities for one part of this workflow; namely, group analyses of the sort described in the prior sections. Here we consider other potential interfaces between DBMS and typical stages in imaging analysis (we follow a typical processing workflow as outlined by Smith, 2002).

The initial stages of image analysis typically involve reconstruction of *k*-space data into functional TS runs. These TS data often undergo a number of transformations before they are analyzed statistically (e.g., alignment, temporal and spatial smoothing, mean adjustment etc.). Because the TS is only analyzed statistically after these steps are completed, there is no strong reason to keep the intermediate data representations in a database as these are rarely needed following preprocessing. They can be stored offline (e.g., on backup tape), or in so-called ‘near-line’ solutions such as relatively slow network-mounted storage repositories.

Whether or not the final TS will be stored in a database depends on the research question. Storing the TS in the database affords convenient execution of sophisticated analyses of TS data such as structural equation modeling (cf., Skipper et al., 2007a, for an example use), and flexible selection of TS subsets on the basis of categorizations of those data (as discussed in the section, An alternative data representation scheme). Yet, oftentimes TS data are not the domain of inquiry *per se*, but are only used for establishing the relative sensitivity of each voxel/vertex to each experimental condition, using standard regression based approaches. Here, there is no strong rationale for storing the entire TS data in a database, but there is good reason to store the signal estimates in each voxel for each experimental condition, as these are the basis for the subsequent second-level group analysis. In any case, the voxels’ coordinates can be stored alongside the statistical values (in the future, this could potentially allow existing command line utilities to interface with database-stored data in the same way they currently operate on flat files).

Importing data from the file representation into the database entails creating a table, and populating it with data from a text file. The following two MySQL commands create a table with three columns, reflecting the assignment of anatomical regions of interest (ROIs) to voxels for each subject, and load data into that table from a text file (vox2roi.txt):

```
create table vox2roi (subject int, voxels int, roi int);
load data local infile 'vox2roi.txt' into table vox2roi fields
terminated by ' ';
```

Database queries can be performed more quickly if the fields (columns) by which data are typically selected have associated ‘indexes’. In this example, it is expected that users would want to select nodes on the basis of some *a priori* ROI classification; in this case, faster data selection could be achieved if the table is created with an index on the ROI column:

```
create table vox2roi (subject int, voxels int, roi int, index (roi)).
```

Once individual data have been registered to common space and stored in the database, group-level analyses of various types can be performed, and the results of such analyses can be stored in the DBMS in the form of information about each voxel.

After group-level statistics have been established for each voxel or surface vertex, they are typically followed by mathematically motivated thresholding procedures. Thresholding controls for the family-wise error (FWE) associated with the multiple statistical tests performed on the data, and with the fact that the data are not independent due to spatial filtering. Spatial filtering is often explicitly introduced in the workflow to increase signal to noise but is also introduced implicitly during any number of spatial transformations of the data, e.g., motion correction, alignment to common space, or volume-to-surface mappings. Some thresholding methods such as random field theory (Worsley et al., 1996) or Monte Carlo simulations of active cluster extent (Forman et al., 1995) estimate the smoothing in the dataset in each axis (i.e., the smoothing kernel specified in terms of full-width half maximum, FWHM), and use this estimate in simulations that establish voxel- or cluster-level thresholds. Currently, these utilities do not operate on database-stored data, and so the estimation of the smoothing kernel and the subsequent clustering could only be performed once the group level results have been converted to a compatible file format. Other thresholding methods, such as those based on permutations (e.g., Nichols and Holmes, 2002) or on false-discovery rate (e.g., Genovese et al., 2002) do not rely on pre-assessment of FWHM. Assessment of FDR is currently available as an “R” package, and permutation methods are easily implemented, and benefit from the capabilities of distributed computing (see Stef-Praun et al., 2007).<sup>2</sup>

Given the importance of being able to visually assess and report the results of imaging analyses (whether in 3D space or cortical surfaces) it is important to know how the results of analyses such as the ones reported here can be graphically displayed. While “R” has graphical output functions, these are quite generic and not customized for the complex display of brain imaging data that often involves visualization of anatomical data and functional overlays. It is also reasonable to assume that researchers would want to display the results of their group- or individual-level analyses in the same space (and interface) from which the input data originated. In some circumstances, the analysis results can be saved and immediately loaded into the graphical interface (e.g., the SUMA software can load single column text files representing

whole-brain activity and display this information directly on a cortical surface image). In other cases, the results of the analyses must be imported to a native file format (e.g., using AFNI’s 3dUndump). There are also two “R” packages specifically aimed at fMRI analysis that can be used to load, save and graphically display anatomical and functional data stored in ANALYZE and AFNI file formats (Marchini, 2002; Polzehl and Tabelow, in press). While we have not used these packages in our data analysis workflow, they offer the future prospect of being able to analyze data in a distributed manner and plot the results from within “R”.

### Clients

In the simplest implementation, both the client and the server can be installed and run on the same machine, whether for purposes of testing or actual data analysis. However, to make full use of the distributed processing capabilities, client software is usually run on a number of computers separate from the host running the database. The client sends a query to the database and receives in return a table (i.e., the set of rows) that satisfies the query (see Appendix for instructions on how to download and invoke an example “R” script that demonstrates this functionality).

### Client implementation

In our approach, clients are implemented in the statistical language “R” (<http://www.r-project.org>), a free, publicly licensed statistical environment similar to the commercial software S/S+ (<http://www.insightful.com>). “R” is compatible with Microsoft Windows and various UNIX based platforms such as Linux or Mac OSX. Similar to other mathematical programming languages, scripts written in the “R” language can access and query relational databases via standard database protocols using SQL.

A simple data analysis script for a cross-participant contrast between two conditions might consist of a small number of steps, e.g.:

- (1) Retrieve data from the database for a certain range of voxels (e.g., voxels numbered 1–100) [SQL Query].
- (2) From the returned data, select the data for the voxel #1 [Internal R array].
- (3) Conduct a statistical test on the data in that voxel (ANOVA, paired sample *t*-test) [Internal R procedure].
- (4) Store the result in a temporary array; select the next voxel (step 2) [Internal R procedure].
- (5) Upon finishing, write the result array to a file [Internal R procedure] or to the database [SQL Query].

The ability to analyze a large number of spatial units also makes DBMS-based approaches applicable to domains such as voxel-based morphometry (Ashburner and Friston, 2000). In such methods, where data are sampled at a high spatial resolution, the number of analysis units can exceed 1.5 million (given in-plane resolutions of  $1 \times 1$  mm or better).

One advantage of using “R” for data analysis is that the retrieved data are directly accessible for examination and manipulation. “R” provides over 600 distinct packages for analyzing and plotting statistical data, covering domains such as Bayesian, multivariate and TS analysis, PCA, ICA, and nonparametric methods. (See the “R” reference manual: <http://cran.r-project.org/doc/manuals/fullrefman.pdf>.) Using these packages we have implemented analyses of fMRI data including (a) standard analysis of variance (ANOVA), (b)

<sup>2</sup> All the thresholding methods mentioned account for spatial smoothing (blurring) in the data. In certain cases, it could be important to spatially filter the data with different smoothing kernels and apply the same analysis to the resulting datasets. In such cases, DBMS offers a convenient way to store multiple versions of individual-level data smoothed with different kernels. These sorts of analyses could be important when it is known that a large smoothing kernel reduces sensitivity to finding activity in certain anatomical regions (Buchsbaum et al., 2005).

clustering of voxels on the basis of Beta values, (c) tests of whether the hemodynamic response peaks at different time points under different experimental conditions, (d) correlations between hemodynamic response functions in different experimental conditions, (e) post hoc contrasts, (f) analyses of functional connectivity, (g) generation of data for permutation tests, (h) voxel-wise correlations between voxel intensities and behavioral data, and (i) reverse correlation methods (see section, Detailed example: reverse correlation analysis).

#### *Client's suitability for distributed computing environments*

The availability of multiple computing nodes holds the promise of speeding up fMRI data analysis by distributing the computational load. For some analytical procedures, such a speed-up is virtually a necessity due to their intensive computational demands. Randomization methods in statistics represent a classic example of combinatorial explosion, and in fMRI analysis, such a procedure is the basis of statistical analyses using permutation tests (e.g., Bullmore et al., 1999; Nichols and Holmes, 2002), in which new datasets are created to assess whether an experimental dataset has characteristics that differ from those found by chance. In such cases the bulk of the analysis is in generating the permutations and performing clustering on each permutation, rather than in running the statistical test itself, making this task optimal for distributed computing. We have shown (Stef-Praun et al., 2007) how permutation-based statistical analysis of fMRI data can be sped up using Grid computing technologies in which multiple computing clusters parallelize both the generation and clustering of permuted datasets.

Client-server based systems are particularly well suited for parallel computing, as the clients are independent of each other, and exploit the availability of computing cycles by breaking up large analysis jobs into smaller jobs and running those jobs simultaneously (parallelizing a single job onto multiple processors can also be implemented, but this issue is outside the current scope; see Li and Rossini, 2001, for more discussion). However, achieving distributed analysis using multiple clients does not necessitate having access to a computing cluster or Grid facilities. At small scales, it is feasible to launch a number of “R” processes on computers in a local laboratory to attain similar functionality.

#### **Detailed example: reverse correlation analysis**

Here we present a detailed implementation (by JIS) of a reverse correlation analysis using the system described above. Reverse correlation is an objective method for associating properties of a stimulus with fluctuations in a TS, in this case with regional fluctuations in the blood oxygenation level-dependent (BOLD) response. In this specific implementation, the database is queried for nodes in given anatomical regions in which activity exceeds a set threshold, and the TS of these nodes is returned from the database for further analysis. The analysis requires that the parcellation of each individual's cortical anatomy into regions has been completed such that each node's data in the database is associated with a symbol (a number) that uniquely identifies an anatomical region. The number of values comprising the TS corresponds to the number of functional brain acquisitions in the study. Within each region, the TSs from the returned nodes are averaged into a single “mean” TS for that region. The fluctuations in the TS are then examined with respect to the timeline of the

stimuli presented in the experiment to evaluate which properties of the stimuli correlate with the signal fluctuations.

#### *Background*

We have shown that the ventral premotor cortex (PMv) plays a role in using observable mouth movements to aid speech perception (Skipper et al., 2007b). The analysis described here examined the impact of observable hand movements on comprehension (detailed results will be reported elsewhere). Participants listened to stories (Aesop's Fables) when a storyteller was either not visible, visible but made no gestures, visible and made meaningful gestures associated with the stories, or visible but made non-meaningful self-adaptive hand movements (e.g., scratching or adjusting clothing). The present analyses tested hypotheses about the effect of hand movements on PMv activity (Skipper et al., 2006). Peaks in the BOLD TS from PMv were predicted to correspond to meaningful gestures when the gestures were visually related to the story content. In contrast, peaks in the TS from PMv were not predicted to correspond to non-meaningful hand movements in these stories. Finally, it was predicted that hand movements would not correspond to peaks in primary auditory and visual cortices.

#### *Data processing steps prior to database import*

Preprocessing stages were conducted prior to loading the data into the database and included: (a) inflating anatomical volumes to a surface representation and aligning them to a template of average curvature using FreeSurfer (Dale et al., 1999; Fischl et al., 1999); (b) automatically parcellating the surface of each participant into anatomical regions using FreeSurfer (Fischl et al., 2004); (c) importing the resulting parcellation into the SUMA software package (Saad et al., 2004); and (d) warping the resulting data to a standard mesh (Argall et al., 2005). Following these steps, all subsequent data analyses were performed on the nodes in the surface domain rather than voxels in the volume domain.

We mapped two types of data from the volume domain to the surface representation (cf. Saad et al., 2004, for details of mapping procedure). These data were: (1) each participant's TS for each voxel (i.e., the signal intensity in a single voxel over time, sampled at each functional image acquisition) and (2) the statistics derived from regression analyses performed on the individual participant data. These latter statistics were obtained by regressing waveforms of the predicted hemodynamic response in each experimental condition against the TS data, thereby establishing the sensitivity of each voxel to each experimental condition. Preprocessing of the raw TS consisted of removing artifactual spikes, removing linear and quadratic trends, and mean normalization. After interpolation to the surface, these two types of data for each hemisphere, for each participant, were imported into separate tables in the database as described in the following section.

#### *Structure of the database*

Given the typical way neuroimaging data are organized within file systems, it is simple to organize data tables in a DBMS so that their structure corresponds to this organization. While creating such analogous structures may not be the optimal configuration for a database schema (as we will discuss subsequently) it is functional and transparent, facilitating use by researchers with relatively little database experience. This was the approach taken here, in the first

database schema design effort by one of the authors (JIS) with extensive experience in file-based fMRI analysis. The database *Gesture* was created in MySQL. This database contained 77 tables (five for each of the 15 participants and two global tables). Specifically, for each of the 15 participants, two tables were associated with each hemisphere: one for the analyzed functional data (beta coefficients) and one for the raw TS data, and the fifth table identified the region associated with each node. Two additional tables contained information relevant to entire group, and stored the baseline values for each participant and which experimental condition was associated with each functional volume acquisition.

### Analysis procedures

“R” was used to carry out the reverse correlation analysis on a computing cluster, utilizing up to 80 computing nodes at a time. The first part of the procedure established representative TSs for the regions of interest (for each condition) and the second part of the procedure performed the reversed correlation analysis. Each computing node was assigned a group of ROIs for analysis (for exploratory purposes, 84 anatomical ROIs were examined in total). The core parallel computation process consisted of repeated database queries that selected, for each participant, the TS of voxels that were reliably active in at least one of the four experimental conditions ( $T=3.32$ ,  $p<0.001$ ). This query was performed for each participant, for each anatomical ROI, in both the left and right hemispheres (given that there were 15 participants and 84 ROIs in each of the two hemispheres, the query was run 2520 times). A specific instantiation of a query (in pseudocode) would be:

```
select all_timeseries_data from participant1_leftHemisphere
Data for surface nodes that are (a) part of ROI_82 and (b)
have a t-value greater than 3.321 in at least one of the four
experimental conditions.
```

Note that this query returns information from the table containing participant 1’s left hemisphere TS data (participant1\_

leftHemisphereData) on the basis of constraints from two different tables: the table assigning nodes to ROIs, and the table storing for each node the  $T$ -valued for the four experimental conditions.

The returned TS data were partitioned (binned) by condition, generating a TS for each of the four conditions. For each such TS, time points with extreme values (signal change  $>10\%$ ) were replaced with the median signal value. For each participant the TS was normalized against the baseline estimation for that participant. Then a mean TS was established for the entire group by averaging over participants. The resulting TSs reflected activity in an ROI during each condition.

Finally, for each TS we automatically identified local maxima and minima in the fluctuating signal and correlated them against the properties of the stimuli presented on the screen (Fig. 3). The TSs were first decomposed by placing gamma functions of variable heights and widths with similarity to the shape of the hemodynamic response at maxima in the TS (grey curves in Fig. 3) as determined by the second derivative of the TS (Rundell, 1990). Half of the full width half maximum (FWHM/2) of the gamma functions determined which of the aligned stimulus attributes were associated with maxima in the hemodynamic response. The distance between the FWHM/2 of two temporally adjacent gamma functions determined which stimulus attributes were associated with minima in the response.

### Results and discussion

We found that in PMv, meaningful gestures resulted in peaks in the TS when those gestures described the content of the stories, and valleys in the TS when the hands were still (Fig. 3A). But, this relationship did not hold for non-meaningful gestures (Fig. 3B). Furthermore, gestures were not associated with peaks in primary auditory or early visual cortex, indicating that the PMv responded to the linguistic meaning and the semantic content in the gestures rather than to lower level acoustic or visual properties of the stimuli. The analysis above was implemented in a distributed manner on a local cluster of 128 nodes (256 processors), in which

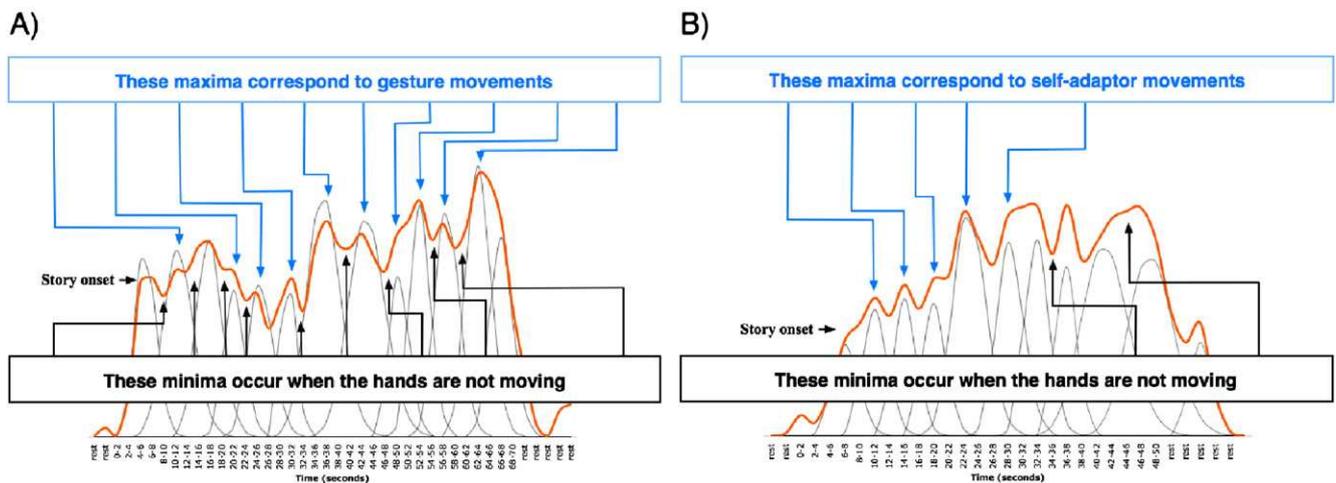


Fig. 3. Results of a reverse correlation analysis performed using a database and Grid computing. Orange lines are the hemodynamic response in the ventral premotor cortex during (A) the gesture condition and (B) the self-adaptor gesture condition, in which gestures were uninformative with respect to story content. Grey lines are the gamma functions fit to each maxima in the response. These were used to objectively determine which stimulus aspects produce maxima and minima (see text). Blue arrowed lines point to maxima while black arrowed lines point to minima. Meaningful gestures were far more likely to occur at maxima in the response than in minima, whereas non-meaningful self-adapting hand movements are as likely to occur at maxima as minima.

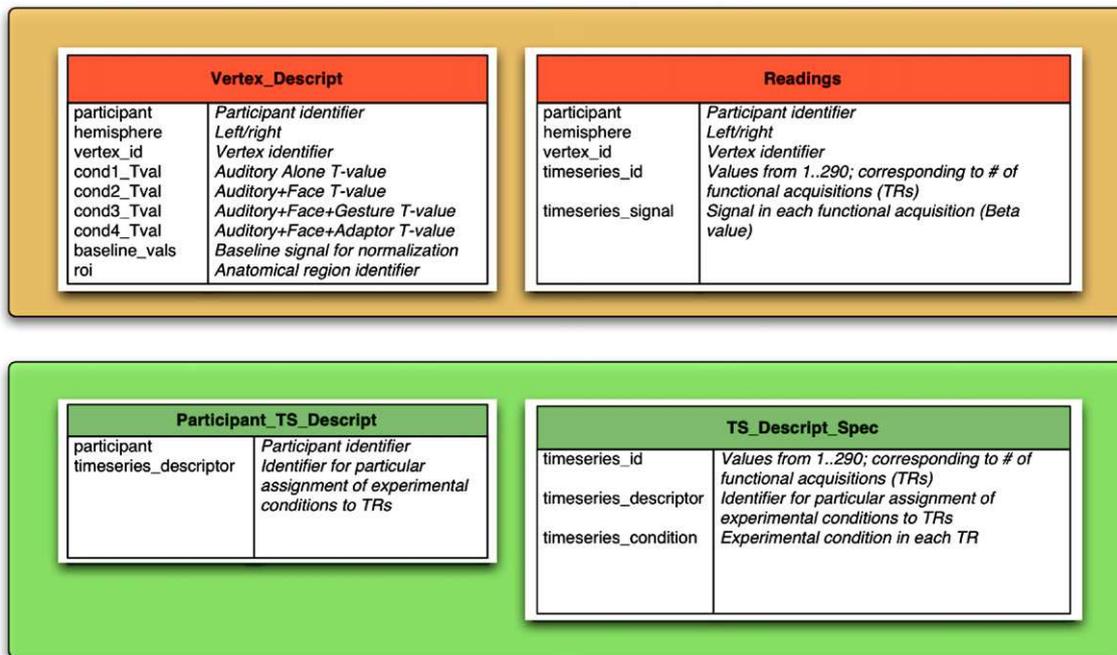


Fig. 4. Example of database schema for storing time series data from an fMRI experiment. The database schema affords selecting the time series of any given set of voxels on the basis of the voxel's estimated signal intensity or anatomical location. In addition, for each voxel it is possible to select either the entire time series, or just those time points in the series where specific experimental condition or conditions occurred.

each cluster node was assigned ROIs for analysis, and the analysis utilized up to 80 computing nodes simultaneously.<sup>3</sup> The speed up afforded by partitioning the job enabled allowed us to understand the results more quickly and consider and devise new questions and hypotheses.

#### An alternative data representation scheme

The database structure outlined in the section Structure of the database includes many tables because each participant's data were assigned a set of 5 tables. This scheme may therefore be impractical for studies involving a large number of subjects. A more efficient scheme can capture the same data in four tables, independent of the number of subjects, and affords queries that have greater or equivalent power (see Fig. 4).

Simple data queries can be performed by using just the upper two tables in the figure (*Vertex\_Descript* and *Readings*). These two tables are sufficient to extract the entire TS of vertices that satisfy functional or anatomical criteria or both (e.g., select the timeseries\_signal of vertices whose cond1\_Tval>5 and whose ROI=5). The lower two tables in the figure allow more sophisticated queries. The first table, *Participant\_TS\_Descript*, marks the trial-order sequence

received by each participant (e.g., some participants would be presented with the auditory-alone condition prior to the auditory-face condition and for others the order was reversed). Each trial-order assignment sequence is marked by a unique identifier in the timeseries\_descriptor field. The second table, *TS\_Descript\_Spec*, specifies the condition presented at each functional acquisition for each trial-order sequence, e.g., whether the first acquired image was associated with the presentation of a meaningful gesture or with a less related adaptor movement. Using the information in these two latter tables, it is possible to extract only those points in the TS associated with a given condition for each vertex or region (e.g., for vertices in ROI=5, select the data acquired when gestures were presented).

This type of scheme is particularly useful for analyses of natural stimuli: for example, we were interested in the types of words or gestures presented during each acquisition. However, descriptors of a TS can also include details such as whether a sentence or phrase has started or ended at a TR or any other descriptor of interest for which a TS subset should be extracted. Indeed, any dimension of interest in the stimuli could be coded. For example, in the domain of vision, one might code the properties of the video frames, such as the amount of visual change between frames. Each stimulus dimension could be resampled to the time scale of the imaging procedure (i.e., the TR was 2 s) and entered into the database as tables. Alternatively, the TS could be resampled to match the stimuli. The fMRI TS could then be mined on a voxel or ROI basis for a relevant stimulus or combination of stimulus dimensions and their correspondences to local maxima or minima in the TS.

#### Discussion

The framework we have described here is one that allows both individual users and larger research centers to store data in a way that

<sup>3</sup> The use of multiple nodes could introduce overhead due to the load on the DBMS. We examined this issue using a representative 10-min group-level analysis job in which each computing node issued two database queries per minute (jobs were executed on a computing cluster at the Argonne National Laboratory, and queried a database at the University of Chicago). The measurements indicated that the time per job remained constant whether 5, 10, 20, 30, 40 or 50 jobs were conducted in parallel. The mean job time per computing node (in seconds+SE) were: 5 jobs: 496 (14); 10 jobs: 460 (12); 20 jobs: 470 (9); 30 jobs: 465 (7); 40 jobs: 479 (8); 50 jobs: 472 (6). Thus, for this representative analysis, use of even 50 computing nodes did not substantially increase overhead.

can be queried efficiently, from both local and remote sites, and that affords distributed statistical analysis of those data. Flexible sharing via ‘view’ mechanisms and flexible security are also inherent features of the system. We have provided details of the server and client implementation, and explained potential interfaces between such DBMS-based systems and other stages of a typical imaging analysis workflow.

#### *Merging distributed computing resources with DBMS for imaging analysis*

The two technologies at the core of this framework are relational database management systems that store data, making them available for remote access, and a distributed computing architecture (cluster or Grid computing) that is used for parallel distributed data analysis. Each technology offers its own distinct advantages, but the strength of the system is in the synergy between the two. DBMS oriented systems do not necessitate large scale distributed computing to aid in imaging analysis. Even when used in a non-distributed setting, the ability to access selected aspects of data from remote locations that is offered by DBMS (e.g., reanalyzing a certain ROI data from abroad) is beneficial to everyday work. Similarly, distributed computing does not ipso facto necessitate DBMS to enable faster statistical analysis. One could construct a framework in which large files are analyzed via distributed computing nodes, with ultimate collation of completed results. One implementation of such a file-based solution would be to propagate the entire dataset to each computing node and implement the types of analyses we have described above using the data selection mechanisms currently offered by command line procedures in imaging analysis packages, and efficient use of ‘mask’ files when necessary (e.g., when using data from other experiments as filtering constraints). Another file-based implementation would be to select just the data needed for any given statistical analysis and propagate those data to the computing nodes in a way that allocates a different part of the dataset to each computing node. This implementation entails a ‘pre-filtering’ step, during which a ‘mask’ of the required data is created by applying a certain filter. In this approach, the requested subset of data is constructed de novo from various flat files in order to optimize each analysis (some imaging analysis software contain functions for optimizing access to large datasets and selecting subsets of the data, e.g., RUMBA’s *librumba*, <http://www.rumba.rutgers.edu/projects.php>). In contrast to such implementations, a DBMS make it possible to set up a single arrangement of the data (i.e., a database scheme) which affords numerous types of queries, while at the same time serving the client with just the subset of the data that is of interest in the specific analysis, and does so without touching the rest of the data. Furthermore, as we have shown, DBMS naturally allows for data selection over networks (e.g., when conducting concurrent analysis of the same data by more than one research center). While specialized file systems can also allow such access, the implementation of network file systems specifically designed for distributed computing is non-trivial. Thus, using DBMS in the context of distributed computing for image analysis affords a relatively easy way for distributed data analysis. As we have outlined here, file-based solutions could potentially afford similar features, but to the best of our knowledge, such schemes have yet to be developed.

#### *Target population*

Who would benefit from storing imaging data using DBMS? On the basis of our experience, two distinct populations could benefit

from such representations. The first are individual researchers for whom DBMS-based storage enables the execution of multiple complex analyses on the same dataset and direct and convenient access to the data. The ability to select highly specific cross sections of data from remote computers over the Internet is also an advantage for this target population, and greatly aids in collaboration and replication. Our experience shows that undergraduates, graduate students and post-doctoral students (without background in computer science), as well as technical staff, can rapidly master the basic syntax of SQL and “R” programming.

The other target population comprises the larger research centers that would likely use the DBMS-based system in the context of a distributed computing environment (whether computing clusters or distributed Grid sites). The framework offers this population a convenient method for storing and sharing data, as well as conducting advanced statistical analyses in a distributed manner. While we have emphasized benefits for analysis of imaging (fMRI, PET) data, the approach described can be extended to researchers interested in other types of data. As we have described, the bulk of database use takes place once those processing tasks more tightly linked to image analysis *per se* have been completed (e.g., filtering, registration, removal of volume acquisitions associated with artifacts). Thus, the analysis of database-stored data could potentially be extended to other types of structural data such as VBM or DTI, once those have been processed with tools specifically dedicated to those types of data.

Finally, we consider the role of new technologies in generating new methods of scientific inquiry in the community, and the likelihood that new target populations would emerge because of the availability of such systems. For example, the ability to analyze easily the same dataset and to share analysis code seamlessly across individuals could foster cooperation between small groups of individuals that transcends the traditional cooperation methods that exist today, and that are based on cooperation between research groups. Thus, one individual could store the data in a DBMS, and 3–4 colleagues would analyze the dataset in parallel pursuing specific and diverging theoretical questions.

#### *Summary*

The increasingly complex research questions addressed by fMRI research impose non-trivial demands on computational infrastructures. Already, these infrastructures need to support management of massive amounts of data in a way that affords rapid and precise data selection, to allow collaborative research, and to do so securely and with minimum management overhead. Here we have presented one approach to overcoming current limitations, which is based on freely available (open source) database management systems that support distributed data analysis using cluster or Grid computing resources. We have described how such a system is practically implemented and have shown via a concrete example the advantages offered by such systems during the analysis of imaging data. Implementing such systems in research centers is likely to facilitate cooperation between research centers and aid researchers in gaining a better understanding of their data.

#### **Acknowledgments**

We thank T. Stef-Praun for his assistance and advice, and two anonymous reviewers for their comments and suggestions. This

work was supported by a grant from the National Institutes of Health: NIH R21-DC008638.

## Appendix A

We have made available an “R” script that can be downloaded and executed locally by individuals interested in evaluating a system of the sort described in the manuscript. When executed, the script will connect to an example database we have set up and conduct some simple queries and statistical analyses. Individuals considering implementing MySQL and “R” may want to download the script and make changes to it. The “R” script and instructions can be found at <http://www.fmri.uchicago.edu/db/db.instructs.html>. Running the script requires installing “R” on the local machine with two packages that enable database access. The website also contains documentation of the job dispatch mechanism described in the Server implementation section.

Users with some experience with Mac OS X or UNIX variants should be able to install R and initialize the script without much problem, following the instructions included on the web address above. However, we do not recommend installing the R client with the database access modules on Microsoft Windows for testing purposes, because installation of the database access package on Microsoft Windows may demand compilation of software on a windows computer (in case the binary package does not install properly), which is somewhat of a lengthy process and requires specialized knowledge.

## References

- Argall, B.D., Saad, Z.S., Beauchamp, M.S., 2005. Simplified intersubject averaging on the cortical surface using SUMA. *Hum. Brain Mapp.* 27, 14–27.
- Ascoli, G.A., 2006. Mobilizing the base of neuroscience data: the case of neuronal morphologies. *Nat. Rev., Neurosci.* 7, 318–324.
- Ashburner, J., Friston, K.J., 2000. Voxel-based morphometry—The methods. *NeuroImage* 11, 805–821.
- Bowden, D.M., Martin, R.F., 1995. NeuroNames brain hierarchy. *NeuroImage* 2 (1), 63–83.
- Buchsbaum, B.R., Olsen, R.K., Koch, P.F., Kohn, P., Kippenhan, J.S., Berman, K.F., 2005. Reading, hearing, and the planum temporale. *NeuroImage* 24, 444–454.
- Bullmore, E.T., Suckling, J., Overmeyer, S., Rabe-Hesketh, S., Taylor, E., Brammer, M.J., 1999. Global, voxel, and cluster tests, by theory and permutation, for a difference between two groups of structural MR images of the brain. *IEEE Trans. Med. Imag.* 18, 32–42.
- Buyya, R., Date, S., Mizuno-Matsumoto, Y., Venugopal, S., Abramson, D., 2005. Neuroscience instrumentation and distributed analysis of brain activity data: a case for eScience on global Grids. *Concurr. Comput.: Pract. Exper.* 17, 1783–1798.
- Dale, A.M., Fischl, B., Sereno, M.I., 1999. Cortical surface-based analysis: I. Segmentation and surface reconstruction. *NeuroImage* 9, 179–194.
- Davidson, S.B., Crabtree, J., Brunk, B.P., Schug, J., Tannen, V., Overton, G.C., et al., 2001. K2/Kleisli and GUS: experiments in integrated access to genomic data sources. *IBM Syst. J.* 40 (2), 512–531.
- Desikan, R.S., Ségonne, F., Fischl, B., Quinn, B.T., Dickerson, B.C., Blacker, D., et al., 2006. An automated labeling system for subdividing the human cerebral cortex on MRI scans into gyral based regions of interest. *NeuroImage* 31, 968–980.
- Eisenberg, A., Kulkarni, K., Melton, J., Michels, J.-E., Zemke, F., 2004. SQL:2003 has been published. *SIGMOD Rec.* 33.
- Fischl, B., Sereno, M.I., Dale, A.M., 1999. Cortical surface-based analysis: II. Inflation, flattening, and a surface-based coordinate system. *NeuroImage* 9, 195–207.
- Fischl, B., Kouwe, A.v.d., Destrieux, C., Halgren, E., Ségonne, F., Salat, D.H., et al., 2004. Automatically parcellating the human cerebral cortex. *Cereb. Cortex* 14, 11–22.
- Forman, S.D., Cohen, J.D., Fitzgerald, M., Eddy, W.F., Mintun, M.A., Noll, D.C., 1995. Improved assessment of significant activation in functional magnetic resonance imaging (fMRI): use of a cluster-size threshold. *Magn. Reson. Med.* 33, 636–647.
- Foster, I., 2005. Service-oriented science. *Science* 308 (5723), 814–817.
- Foster, I., Kesselman, C., Tuecke, S., 2001. The anatomy of the Grid: enabling scalable virtual organizations. *Int. J. High Perform. Comput. Appl.* 15, 200–222.
- Fox, P.T., Lancaster, J.L., 2002. Mapping context and content: the BrainMap model. *Nat. Rev., Neurosci.* 3 (4), 319–321.
- Gardner, D., et al., 2003. Towards effective and rewarding data sharing. *Neuroinform. J.* 1, 289–295.
- Genovese, C.R., Lazar, N.A., Nichols, T., 2002. Thresholding of statistical maps in functional neuroimaging using the false discovery rate. *NeuroImage* 15 (4), 870–878.
- Gray, J., Liu, D.T., Nieto-Santesteban, M., Szalay, A., Heber, G., DeWitt, D.J., 2005. Scientific data management in the coming decade. *ACM SIGMOD Rec.* 34 (4), 34–41.
- Keator, D.B., Gadge, S., Grethe, J.S., Taylor, D.V., Potkin, S.G., FIRST BIRN, 2006. General XML schema and associated SPM toolbox for storage and retrieval of neuro-imaging results and anatomical labels. *Neuroinformatics* 4, 199–212.
- Kotter, R., 2001. Neuroscience databases: tools for exploring brain structure-function relationships. *Philos. Trans. R. Soc. Lond., B Biol. Sci.* 356, 1111–1120.
- Li, N., Rossini, A., 2001. RPVM: cluster statistical computing in R [Electronic Version]. *R. News* 13, 4–7 (URL: <http://CRAN.R-project.org/doc/Rnews/>).
- Marchini, J., 2002. AnalyzeFMRI: an R package for the exploration and analysis of MRI and fMRI datasets. *R. News* 2, 17–24 (URL: <http://CRAN.R-project.org/doc/Rnews/>).
- Nichols, T.E., Holmes, A.P., 2002. Nonparametric permutation tests for functional neuroimaging: a primer with examples. *Hum. Brain Mapp.* 15, 1–25.
- Nieto-Santesteban, M.A., Szalay, A.S., Thakar, A.R., O’Mullane, W.J., Gray, J., Annis, J., 2005. When database systems meet the grid. Paper Presented at the CIDR 2005.
- Patterson, R.D., Uppenkamp, S., Johnsrude, I.S., Griffiths, T.D., 2002. The processing of temporal pitch and melody information in auditory cortex. *Neuron* 36, 767–776.
- Pfister, G.F., 1998. In Search of Clusters: Prentice-Hall, Inc. Upper Saddle River, NJ.
- Polzehl, J., Tabelow, K., in press. fMRI: a package for analyzing fmri data. *R News*.
- R Development Core Team, 2005. R: A Language and Environment for Statistical Computing, Vienna, Austria.
- Rundell, G., 1990. Peakfit. Non-linear Curve Fitting Software. Jandel Scientific, San Rafael, USA.
- Saad, Z.S., Reynolds, R.C., Argall, B., Japee, S., Cox, R.W., 2004. SUMA: an interface for surface-based intra- and inter-subject analysis with AFNI. Paper Presented at the Proceedings of the 2004 IEEE International Symposium on Biomedical Imaging.
- Skipper, J.I., Nusbaum, H.C., Josse, G., Goldin-Meadow, S., Small, S.L., 2006. The cortical motor system simulates action descriptions conveyed by words and gestures. Paper Presented at The Annual Meeting of the Cognitive Neuroscience Society, San Francisco, CA.
- Skipper, J.I., Goldin-Meadow, S., Nusbaum, H.C., Small, S.L., 2007a. Speech associated gestures, Broca’s area, and the human mirror system. *Brain Lang.* 101, 260–277.
- Skipper, J.I., van Wassenhove, V., Nusbaum, H.C., Small, S.L., 2007b. Hearing lips and seeing voices: how cortical areas supporting speech production mediate audiovisual speech perception. *Cereb. Cortex*, doi:10.1093/cercor/bhl147 (Advance Access published on January 19, 2007).
- Smith, S.M., 2002. Preparing fMRI data for statistical analysis. In: Jezzard,

- P., Matthews, P.M., Smith, S.M. (Eds.), *Functional MRI: An Introduction to Methods*. Oxford University Press, New York.
- Smith, K., Jajodia, S., Swarup, V., Hoyt, J., Hamilton, G., Faatz, D., et al., 2004. Enabling the sharing of neuroimaging data through well-defined intermediate levels of visibility. *NeuroImage* 22, 1646–1656.
- Stef-Praun, T., Foster, I., Hasson, U., Hategan, M., Small, S.L., Wilde, M., 2007. Accelerating medical research using the Swift Workflow System. Paper Presented at the HealthGrid 2007, Geneva.
- Stoeckert, C.J., 2005. Functional genomics databases on the web. *Cell. Microbiol.* 7, 1053–1059.
- Szalay, A., Gray, J., 2006. 2020 Computing: science in an exponential world. *Nature* 440 (7083), 413–414.
- Temal, L., Lando, P., Gibaud, B., Dojat, M., Kassel, G., Lapujade, A., 2006. OntoNeuroBase: a multi-layered application ontology in neuroimaging. Paper Presented at the Formal Ontology meets Industry (FOMI).
- Van Horn, J.D., Grethe, J.S., Kostelec, P., Woodward, J.B., Aslam, J.A., Rus, D., et al., 2001. The Functional Magnetic Resonance Imaging Data Center (fMRIDC): the challenges and rewards of large-scale databasing of neuroimaging studies. *Philos. Trans. R. Soc. Lond., B Biol. Sci.* 356, 1323–1339.
- Wagner, A.D., Schacter, D.L., Rotte, M., Koutstaal, W., Maril, A., Dale, A.M., et al., 1998. Building memories: remembering and forgetting of verbal experiences as predicted by brain activity. *Science* 281, 1188–1191.
- Worsley, K.J., Marrett, S., Neelin, P., Vandal, A.C., Friston, K.J., Evans, A.C., 1996. A unified statistical approach for determining significant signals in images of cerebral activation. *Hum. Brain Mapp.* 4 (1), 58–73.