Details about System Specifications

Frank E Harrell Jr
Aaron Mackey

January 12, 2002

1 Choice of PostgreSQL over MySQL

The follow quotations are from Doug Bates of the University of Wisconsin, who has used both MySQL and PostgreSQL extensively.

> I have found that MySQL is easier to set up and to administer than is PostgreSQL but the added flexibility in PostgreSQL more than makes up for its somewhat unwieldy nature.
>
> Views are surprisingly important to me. One large database that I work with consists of records from the Texas Assessment of Academic Skills (TAAS) with about 11 million records of math test results. The original records have (inconsistent) information on students and on schools and school districts embedded in each test score record. A big part of the data cleaning was extracting consistent information wherever we could. That gives us tables of test results that have identifying numbers for student, school, and school district. Those identifying numbers reference the student table, the school table and the district table. In MySQL I have to use explicit joins to sew this information back together if, for example, I want to use sex or ethnicity of student in a model of the test scores. In PostgreSQL I can create and store a view and treat it as a single table. The view looks like a table but is really a virtual table. It just stores the recipe of how to relate test results to student information to school information and school district information so the whole thing appears to be one big table. This may seem trivial but the view has two advantages over using explicit joins each time you access the information:
>
> - The expression defining the join is only entered once. This saves on typing but, more importantly, it saves on errors. If you have ever made the mistake of omitting a condition in a join and accidently creating an outer join when you meant to create an inner join, this becomes important. The CVS server

for the R project was brought down last year by a student who made this mistake in MySQL.

- the query optimizer can examine the join in detail and work out an optimal plan.

If you think of it, a factor in the S language is a specialized type of view so we are familiar with the concept.

PostgreSQL is extensible. You can define your own functions using SQL or PL/PGSQL or Perl or Python. You can also define new data types. For the Current Index to Statistics we have specialized data types for ISSN and ISBN numbers and special operators and functions for these data types.

I would find it very confining to go back to MySQL these days but I don't begrudge people who use MySQL their choice. In a way the MySQL/PostgreSQL religious wars are very much like the vi/emacs religious wars. MySQL, like vi, is smaller and faster for the standard tasks. PostgreSQL, like emacs, is hard to adjust to initially but its flexibility and extensiblity make the steeper learning curve worthwhile.

I was just creating a PostgreSQL database and realized another advantage of PostgreSQL - in PostgreSQL tables you can use external keys to enforce referential integrity. In the example I was describing where the records in the test score table include a student identifier and a school identifier and a district identifier, I can indicate in the definition of the table that the student id refers to the primary key in the student table. An attempt to insert a record into the test score table with a student id that does not occur in the student table will produce an error.

Duncan Temple Lang of Bell Labs added the following.

The REmbeddedPostgres package allows one to use R functions directly in Postgres. While we have also done this in MySQL it is not in any way supported by MySQL and requires some modifications to the system. For Postgres, this is a straightforward extension that follows the Postgres documentation! Similarly, we may well be able to use data types in R as elements in Postgres tables in the future.

Personally, I think that as this sort of embedding within sophisticated and well-tuned database management systems becomes more secure and stable, we will be able to handle significantly larger datasets with R code.

2 Screen Layout

See `www.humanfactors.com/layout`.

The system will create a default HTML layout (basically a long layout with one column of data). The system will autogenerate template files (without all the extraneous javascript and navigational elements) at the users request. These can then be hand-edited and the system made aware of which template file to use (allowing you to have multiple template files to "try out" during development).

This will be the usual web-browser file download/upload circuit (i.e. the system will provide you with either the default template, or some other template that's already been stored - you can edit it locally, then upload it again as either a new template, or a replacement for an already existing template; separately, you define for each CRF what template should be used in it's display (from the list of available, stored templates).

The template format will probably have very simple "placeholder wrappers" for the data elements - e.g.:

```
<html>
<!-- The "template" -->
<table>
  <tr>
    <td><!-- FIELD NAME="name" ATTR="prompt" -->Name<!-- /FIELD -->:</td>
    <td><!-- DATAENTRY NAME="name" SIZE="50" --><input type="text" name="name"
         maxlen="128"><!-- /DATAENTRY --></td>
  </tr>
  <tr>
    <td><!-- FIELD NAME="sex" ATTR="prompt" -->Sex<!-- /FIELD -->:</td>
    <td>
      <!-- DATAENTRY NAME="sex" SIZE="1" -->
      <select name="sex">
        <option value="">Choose:</option>
        <option value="M">Male</option>
        <option value="F">Female</option>
      </select>
      <!-- /DATAENTRY -->
    </td>
  </tr>
</table>
</html>
```

The idea here is that the template be valid html that when viewed in a browser (or HTML editor), it looks much like the final product; The "templateness" of it resides in the HTML comments - each dynamically generated element (the prompt or the data entry element) is surrounded by XML-esque tags which themselves are embedded within HTML comments (so they're not visible in the browser "preview"). Content between the XML template tags is only for previewing, it doesn't ever get interpreted (only the instructions contained within the XML tags get interpreted). This way you can move around elements (including the surrounding XML tags), and control the display of elements (via attributes in the XML tag), and preview the result as well. And you can add any other extra HTML (¡hr¿, embedded tables, etc) you like.

Note in the example above that the "name" input field doesn't have a size attribute, while the XML instruction does; here's the final output after using the template above:

```
<html>
<!-- The "real" output -->
<table>
  <tr>
    <td>Name:</td>
    <td><input type="text" name="name" maxlen="128" size="50"></td>
  </tr>
  <tr>
    <td>Sex:</td>
    <td>
      <select name="sex" size="1">
        <option value="">Choose:</option>
        <option value="M">Male</option>
        <option value="F">Female</option>
      </select>
    </td>
  </tr>
</table>
</html>
```

If a field is later added to a form definition, but no corresponding label exists in the template, a warning can be issued, and the field added to the bottom of the current template (in the default tabular format). For the reverse situation, when a field is removed, the template position for that field will just get filled in with ` `; blank space (removing it entirely could possibly break table structure, don't want to have to start parsing all the HTML to do it right).

The user needs to be able to easily navigate logical sections of data. These sections might be inherited from the XML metadata such as Data Collection Modules. The user should be able to put any sectioning commands she wishes in the editable html template described above. Such sections may even represent page numbers on a paper form. Section names should appear on a navigation panel on the data entry screen, probably as a pull-down menu.

The system will automatically generate a navigation bar to appear on a panel on each screen, for navigating by data collection module (DCM) if any DCMs were used. Default HTML will include a horizontal rule before a new DCM begins. The data manager can also hand-edit anchors and tags in the HTML that allow for non-database-driven navigation of data screens (e.g., by page numbers on printed versions of case report forms). The system will scan these anchors and generate a second navigation bar.

3 Prompts for Field Entry

The default prompt to be inserted in HTML for user entry of a field will be the field label. A customized field prompt can be specified in the field metadata.

Fields for which the URL suffix exists in the field metadata will have the field prompt hyperlinked to the complete URL formed by combining the study base URL and this suffix.

4 Entry Formats for Specific Data Types

At login time users can select formats for entering certain field types. Dates may be entered as `mm/dd/yyyy`, `dd/mm/yyyy`, or `yyyy/mm/dd`. Two digit years will automatically have 19 or 20 placed before them (19 if the two digits are greater than the current year, 20 if the two digits are at or before the current year). That is, as of 2001 a 20 will be placed in front of a 2-digit on the screen if the 2 digits are 00 or 01.

Times will be entered as either 24-hour time, or as `hh:mmx` where `x` is either `a` or `p`.

5 Check Boxes, Pull-down Lists, and Table Look-up

Multiple-choice fields should have check boxes for each choice. Single-choice fields should use pull-down lists.

For table look-up fields the field metadata specifies three new elements: a table name, the name of the field to look-up, and the name of the field to store as the field value (if different from the look-up field). The look-up field will drive selection during data entry. The user enters the first letter of a string and the first entry in the look-up table whose first letter matches that is displayed. The user enters the second letter and the first string matching the first two letters is displayed. This process continues in like fashion until the user has uniquely identified the string to use.

A complication arises when returning to a record, if the optional "value to store" was used. This may need back-conversion to the look-up value. It may be best if to the right of the field prompt appeared both the look-up and stored value even though only one of these is stored.

A different way for handling all this may be preferred. Whether single, multiple choice, or look-up, either let users specify display format at login time, or use the following scheme:

1. Always use check boxes if there are fewer than 12 possible choices
2. Use a pull-down list for 13-40 possible choices if only a single choice is allowed
3. Use a pull-down list with highlighting of one option and promotion of that selection to an accumulation of k selections to the right (after $\rightarrow$), where k is defined in the field or choice metadata. Demotion (with $\leftarrow$) for removal of choices should also be allowed.

4. For more than 40 possible choices, use alphabetic-partial match-choice completion. For multiple choice fields the "promotion to accumulated choices" would be used in conjunction with this.

6 Check Digits

There should be an option to require a LUHN check digit check upon entry of certain ID variables. For a definition see `whatis.com` and search for `LUHN formula`. The is the formula used by all major credit card companies.

7 Tool for Field Validation

The Perl `Data::FormValidator` module will be used for server-side form validation. See `http://search.cpan.org/doc/MARKSTOS/Data-FormValidator-1.7/lib/Data/FormValidator.pm`.

8 Form and Record Navigation

At the bottom of any browser screen need arrows that when clicked bring up previous or next record in the currently displayed CRF. It would also be good to have symbols to click to get to the beginning or end of the entire file.

Click sites for `Add`, `Update`, `Delete Record` are needed.

Tabs or similar selection areas are needed to cause the main screen to be replaced with any chosen CRF for the same primary key value to be displayed

When subjects can have multiple records for a CRF a certain click by the user should pull up a chronological list of all records for the subject for that CRF. By clicking on any one of these the user should be taken to that instance.

At login time users can select an option specifying whether by default to display the last record entered at the site, the first record, or a management page.

9 Client-Side Scripting

A generic Javascript function for basic checking of numeric data could be invoked as follows.

```
chk.numeric(x, type, mandatory=T, hmin, hmax, smin, smax)
```

`x` is the field contents, `hmin` and `hmax` are hard limits (error message, clear field on screen if violated), `smin` and `smax` are soft limits (warning message, containing something like `Click here to confirm entry of the value`. Messages could go to a separate frame which is always in the same place on the screen (e.g., bottom left). Any or all of the 4 limits could be missing, meaning no limit. Messages should say e.g. `field` $>$ `hmax`, `field` $<$ `smin` to tell the user exactly

what the violation was and what the limit was defined as. The `type` argument might specify `integer` or `float` so that if `integer` a check for a decimal place could be made. If `mandatory=T` the user is forced to enter a value for that field.

Corresponding Perl functions for server-side checking would be good to have too.

For table look-up fields, the table of look-up values and optional stored values would be queries at the beginning of a session and converted to Javascript vectors for fast lookup.

10 Derived Variables

Derived variables will not be stored in the database. They will appear on the data entry screen as read-only fields after being calculated swiftly. They will be re-derived as needed during SQL queries of the data. One good way to handle derived variables is through the `REmbeddedPostgres` package (see `http://www.omegahat.org/RSPostgres/`) by which R code can be executed in PostgreSQL procedures.

11 Administration Screen

Users should be able to call up a screen that lists all the forms that have been entered for a subject, with the date of data collection and the date the data were last modified. The percentage of missing fields on each form (for non-mandatory fields) should be displayed.

12 Data Views

Except for sub-tables, data will be viewed and entered in form format with one subject per screen. There needs to be an option to display (but not change?) data on multiple subjects in a tabular format.

13 Security

All communications are encrypted using modern SSL technology and strong encryption. Servers will employ pass-phrase encoded encryption keys such that a successful server breakin will not compromise data security. Database access will be password protected; the password(s) for server access to the database will also be encrypted within the server's memory space, using the aforementioned passphrase encryption method. All this jargon can be summed up as: there are no unencrypted, non-password protected files that can be read by an intruder. Both communications and data storage are protected from eavesdropping and intrusion, independently of any efforts made to secure the machine itself. As a

side effect, if the server goes down, it must be brought up manually by a human (so that the critical passphrase may be provided).

We assume that users' browsers have strong encryption turned on.

User access will time out after 30 minutes of inactivity, to prevent unauthorized access if the user leaves the station.

A `robot.txt` file will be set up on the web site so that search engines do not find the server site.

A software firewall (ipchains in Linux) will be set so that access is only available for a list of IP addresses corresponding to study personnel. For at-home access, users will have to use a proxy server their institution.

13.1 User Roles for Security

Start with the basic dichotomy between "Administrator" and "Project-specific Usage". I can guess that under "Project-specific Usage" would come various sub-roles such as "data entry only", "data review", "data analysis", etc. But I want to try to keep a distinction between "Privileges" (ability to add/view/edit/delete a form/field/data value) and "Roles" (I am a "well-trained data entry clerk" and therefore I have add/view/edit privileges on my projects' data values).

So, to begin:

13.1.1 Role: Super User

Privileges:

- (ALL)

13.1.2 Privileges for Role of Project-specific Super User

- access to all projects assigned to user
- access to all forms assigned to project
- access to all fields assigned to forms
- ability to view all existing patient records of project
- ability to add new patient records of project
- ability to edit all existing patient records of project
- ability to delete all existing patient records of project
- ability to view all private patient data from all records of project
- ability to edit all patient data from all records of project

13.1.3 Privileges for Role of Project-specific Data Entry User

- access to all projects assigned to user
- access to all forms assigned to project
- access to all fields assigned to forms
- ability to view all existing patient records of project
- ability to add new patient records of project
- ability to edit all existing patient records of project
- ability to edit all patient data from all records of project

13.1.4 Privileges for Role of Project-specific Access

- access to all projects assigned to user
- (no other privileges - need to be added manually)

Remember that the proposed design allows a very flexible combination of privileges on projects vs. forms vs. fields. e.g., a limited data-entry user could have access to only 1 form, in which they can only view some data (other data being XXX-ed out), and only edit/update others. The idea of "Roles" is to make a generic set of privileges (associated with the role) that can be given to people "in batch".

Remember that the basic set of privileges are access/view/add/edit/delete. Each of these privileges has meaning when referring to a project, patient record, form, or field.

Privilege	Object	Meaning
view	project	can see the project in main project listing
access	project	can gain access to the project from the main listing
add	project	can create a new project
edit	project	can edit a project's definition
delete	project	can delete a project
view	record	can see listings of patient records
access	record	can gain access to specific patient records
add	record	can add a new patient record
edit	record	can edit a patient record
delete	record	can delete a patient record[a]
view	form	can see the form in the form listing
access	form	can access data for this form
add	form	can add new patient data for this form
edit	form	can edit patient data within this form
delete	form	can delete patient data within this form
view	field	can see this field in the form display
access	field	can see the value (otherwise XXXX'ed out)
add	field	can enter new data in a field
edit	field	can change existing data in a field
delete	field	can remove data from field

[a]Record privileges may also be keyed off of user site/patient site, i.e. a user can have record-level access to patients seen at their own site, but not at other sites.

13.2 Masking of Coordinating Center to Subject Identifiers

Certain multi-center projects need to have patient identifying information collected locally that is not available to the data center. This is handled by encrypting those fields using a passphrase/key supplied by the user. The key is not stored in the database. Each remote site will be responsible for remembering their (single) passphrase and for seeing that all users from that site who should have access to subject identifiers know the passphrase.

The passphrase is entered at signon and will stay in effect for all forms used during that session. At signon, the user has to enter the passphrase twice, and they don't match the user has to enter it twice again until the two match. The system will fail if the user has already entered data and enters the wrong passphrase twice.

14 Accessibility of Server

We are planning on using a dedicated machine for the server, so that accessibility problems would normally result only from internet traffic problems (rare, but they happen). Local data storage brings up security and data management/integrity concerns, but can be accomplished if felt to be necessary.

15 User Actions for Data Transmission

Typically, data on a page is sent "manually", by the user interacting with a page element (clicking a button or link). However, it is quite possible to save each data entry element as it is completed, using a hidden "data pipe". This will require a modern browser with JavaScript support turned on to function correctly.

If the "immediate data save" feature mentioned above is enabled, we can switch the color coding in "real-time" as data is entered. The current plan is to require the user to hit a `Save` button to save the values in the currently viewed record.

16 Restricting Access According to Site

Varying levels of data security (both viewing and editing privileges) may be assigned for each field in the form; e.g. a user may be given viewing privileges for patient study number, editing privileges for specific lab data, but no privileges (viewing nor editing) for personal identifying data. The default privileges for any new user are none - privileges to view and/or edit data must be granted to users administratively.

17 Audit Trail

The audit trail will be generated from Perl; we will not use the default PostgreSQL audit trail. A relational scheme for storing the audit trail could be as follows.

Table `data`

- `datum_id` (primary key)
- `subject_id` (references "subjects" table)
- `field_id` (references "fields" table)
- `value` (polymorphic type for numbers, string, enum, etc)

Table `field_update_log`

- `log_id` (primary key)
- `timestamp`
- `user_id` (references "users" table)
- `data_id` (references "data" table)
- `status` (enum 'new', 'edit', 'delete', 'clear')

- `previous_value` (default NULL, another polymorphic type);
- `new_value` (could be NULL for delete/clear options, polymorphic type).

In this way the log remains compact, but references the necessary info (the foreign key points to the data element actually being changed, which in turn has all the subject/field information required).

References

[1] C. A. Brandth, P. Nadkarni, L. Marenco, B. T. Karras, C. Lu, L. Schacter, J. M. Fisk, and P. L. Miller. Reengineering a database for clinical trials management: Lessons for system architects. *Controlled Clinical Trials*, 21:440–461, 2000.

[2] D. Conway. *Object Oriented Perl*. Manning, 1999.

[3] R. M. Curley, R. L. Evans, J. Kaylor, R. M. Pogash, and V. M. Chinchilli. Development and deployment of an Internet-based data management system for use by the Asthma Clinical Research Network. *Controlled Clinical Trials*, 22:135S–155S, 2001.

[4] S. J. Kunselman, T. J. Armstrong, T. B. Britton, and P. E. Forand. Implementing randomization procedures in the Asthma Clinical Research Network. *Controlled Clinical Trials*, 22:181S–195S, 2001.

[5] W. W. Marshall and R. W. Haley. Use of a secure internet Web site for collaborative medical research. *Journal of the American Medical Association*, 284:1843–1849, 2000.

[6] R. M. Pogash, S. J. Boehmer, P. E. Forand, A. Dyer, and S. J. Kunselman. Data management procedures in the Asthma Clinical Research Network. *Controlled Clinical Trials*, 22:168S–180S, 2001.

[7] L. Stein. *Writing Apache Modules with Perl and C: The Apache API and `mod_perl`*. O'Reilly, 1999.