CENTRALIZED COLLECTION OF EXPERIMENTAL DATA IN AN ONLINE DATABASE

James W. Jordan

A thesis submitted to the faculty of the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Master of Science in the Department of Biomedical Engineering.

Chapel Hill
2007

Approved By:
Dr. Mark Tommerdahl
Dr. Robert Dennis
Dr. Shawn Gomez
ABSTRACT

JAMES JORDAN: Centralized Collection of Experimental Data in an Online Database
(Under the direction of Dr. Mark Tommerdahl)

Unique quantitative sensory testing methods and associated testing apparatus developed by Tommerdahl et. al. have enabled objective evaluation of intra-cortical communication mechanisms, which appears to be sensitive to neurologic disorders that impair normal sensory function. The proposed non-invasive diagnostic procedure enabling objective assessment of the status of the glia in sensory cortex of a patient has the potential for significant impact in the fields of gerontology, toxicology, and general medicine in the way of autism and chronic pain diagnosis, and acute head injury assessment. A central aim of this project is efficient storage of data obtained from disparate research locations into a centralized database, accessible to all researchers utilizing this same testing method and apparatus. Due to its aggregate nature of many different data sources, this ontological database will allow for invaluable data mining across a growing population of subject and patient observations.
## TABLE OF CONTENTS

**LIST OF FIGURES** ............................................................................................................ v

I. Introduction ......................................................................................................................... 1

II. Specific Aims ....................................................................................................................... 2

III. Database ............................................................................................................................ 4

IV. Web Interface .................................................................................................................... 6  
   a. Login Page .................................................................................................................... 6  
   b. Admin / User Central Page .......................................................................................... 8  
   c. User Details Information .............................................................................................. 9  
   d. Experiment Details Page ............................................................................................ 10  
   e. Quick Launch Functions ............................................................................................. 14  
   f. Experimenter Details Page ......................................................................................... 18  
   g. Subject Details Page ..................................................................................................... 23  
   h. Type Details Page ......................................................................................................... 29  
   i. Protocol Details Page ................................................................................................... 33  
   j. Query Builder Page ....................................................................................................... 38  
   k. Account Information Page ........................................................................................... 40  
   l. About Us Page ............................................................................................................... 40  

V. Impact and Future Direction ............................................................................................... 43
LIST OF FIGURES

Figure 1 .................................................................................................................................. 6
Figure 2 .................................................................................................................................. 8
Figure 3 .................................................................................................................................. 10
Figure 4 .................................................................................................................................. 11
Figure 5 .................................................................................................................................. 13
Figure 6 .................................................................................................................................. 15
Figure 7 .................................................................................................................................. 18
Figure 8 .................................................................................................................................. 19
Figure 9 .................................................................................................................................. 20
Figure 10 ................................................................................................................................ 22
Figure 11 ................................................................................................................................ 23
Figure 12 ................................................................................................................................ 25
Figure 13 ................................................................................................................................ 26
Figure 14 ................................................................................................................................ 28
Figure 15 ................................................................................................................................ 29
Figure 16 ................................................................................................................................ 30
Figure 17 ................................................................................................................................ 31
Figure 18 ................................................................................................................................ 32
Figure 19 ................................................................................................................................ 33
Figure 20 ................................................................................................................................ 35
Figure 21 ................................................................................................................................ 36
Figure 22 ................................................................................................................................ 37
Figure 23 ........................................................................................................................... 38
Figure 24 ........................................................................................................................... 41
I. Introduction

Unique quantitative sensory testing methods, based on information obtained from neurophysiological studies of the nonhuman primate cerebral sensory cortical response to a variety of modes of natural skin stimulation, have been recently developed (Tannan V, 2006; Tannan V D. R., 2005; Tannan V D. R., 2005; Tommerdahl M, 2007). These methods and associated testing apparatus have enabled objective evaluation of intra-cortical communication mechanisms, which appears to be sensitive to neurologic disorders that impair normal sensory function. The proposed non-invasive diagnostic procedure enabling objective assessment of the status of the glia in sensory cortex of a patient has the potential for significant impact in the fields of gerontology, toxicology, and general medicine in the way of autism and chronic pain diagnosis, and acute head injury assessment. The testing method has many additional strengths: 1) the device is portable and reproducible; 2) the testing protocols are fast; 3) the protocols are easy for a patient to understand; 4) the method is non-invasive; 5) the tests developed to date have a rapidly growing subject database for comparison. A central aim of this project is efficient storage of data obtained from disparate research locations into a centralized database, accessible to all researchers utilizing this same testing method and apparatus. Due to its aggregate nature of many different data sources, this ontological database will allow for invaluable data mining across a growing population of subject and patient observations.
II. Specific Aims

Multiple decades of research in systems neuroscience has yielded the finding that implicates the role of cortical-cortical interactions in sensory perception. More specifically, preliminary findings suggest that neurological diseases/disorders such as autism, schizophrenia, Alzheimer’s, ADHD, chronic pain, TBI, and migraine show altered cortical functionality which can be detected with dynamic sensory testing. This testing consists of utilizing two vibrotactile stimulus points on the skin and then altering the distance between these points, the stimulus frequency spectrum, amplitude, and phase of both points or each point independently. These alterations allow the assessment of the intra-cortical functional connectivity between well-established regions in primary sensory cortex. The mechanistic hypothesis which forms the basis of the testing method is a result of preliminary data on human subjects combined with *in vitro* direct cortical stimulation of brain slices and *in vivo* neurophysiological signals evoked by natural skin stimulation in animal studies. The resulting testing method and vibrotactile stimulus apparatus is non-invasive, quick to administer, and easy for both patient and diagnostician to understand. The apparatus is also designed for rapid manufacture and ready deployment to a large number of researchers. The use of the device – currently at 10 units - has led to a rapidly growing need for a subject database which can be utilized by both current and future researchers.
The current research project grew out of an original desire to modernize a tactile sensory testing system for basic human psychophysical research. The expanded capabilities of the new system would make a large number of experiments very accessible which had previously been difficult, if not impossible, to perform. This was largely due to the fact that previous sensory diagnostic equipment focused on the peripheral nervous system, and thus assessments of cerebral cortical information processing required lengthy protocols (on the order of 1-3 hours) to be administered in a laboratory setting. The new testing system reduces these cerebral cortical functionality protocols to 1-5 minutes.

This new system, combined with the research knowledge that central cortical processing had larger diagnostic implications than the established peripheral focus, led the research team to begin focusing on collaborative efforts in a number of basic research and clinical venues in order to obtain quantitative measures of changes in cerebral cortical function under a number of varying neurological conditions. The long range goal of the project is to evolve the current diagnostic system from a research tool to a stand-alone device with a wide range of clinical applications.

Two of the research project goals are:

1. Develop a software database system that will make multiple types of clinical research studies feasible (longitudinal, pharmacological, genotype/phenotype, demographic, etc.).

2. Develop user training materials to primary health care workers to train with and effectively employ this device in their clinical environment.
III. Database

The database utilizes the LAMP architecture: Linux, Apache, MySQL and PHP. Linux and Apache provide the operating system (OS) and web server framework, though the database and associated functions are designed to run on any OS implementation of MySQL and PHP. MySQL is an open-source enterprise-class database alternative to expensive SQL server offerings from Microsoft and Oracle. PHP is a web coding language that enables dynamic input and output from the database. The web front-end provides function both to the database administrators and the experimenters who will utilize the database in the course of their experiments. For administrators, abilities include adding/removing information from key tables, registering new experimenters, obtaining statistics on all experiments and registration details of experimenters, and a dynamic SQL query builder which allows a virtually limitless amount of information to be pulled from the database in real-time. For regular users, abilities are reduced, and include adding/removing/modifying associated subject information, adding/viewing associated experiments and data, and adding/viewing experimental protocols. Access restrictions are designed so that experimenters are only able to access their own research information, while some of the information, such as experimental protocols and types, are shared by all registered experimenters. Additionally, results from subjects with a neurological assessment of normal are considered control results, and are thus available for comparative analysis across all experimenters. The true collaborative power lies
within this analytical capability, as it instantly provides experimenters with a large set of control subject data associated with the particular metric being researched. Toward this end, the database itself is relational, allowing data to be both broken into manageable chunks while remaining associated with the original experimenter. This structure permits the aforementioned access control of restricting certain items such as individual experiment data and subject information, while sharing other information such as experimental protocols across all experimenters.
IV. Web Interface

a. Login Page

![Cortical Database Login](image)

**Usage Instructions**

A valid login is necessary to access the majority of the information on the Cortical Metrics website. The only information accessible without an account is the “About Us” section of the site. All attempts to access pages directly either without logging in, or after a user’s session has expired will redirect the user to the login page. A link is provided on this page for registration, in the event that the user has not signed up for an account previously. The user must enter a valid “Username” and “Password” and then click “Submit” to gain entry to the Cortical Metrics site. Additionally, this page displays the total number of experiments contained in the Cortical Metrics database for marketing purposes.
**Development Notes**

Every function in the Cortical Metrics site which requires a valid login first calls the `check_login` function, located in the `functions.php` file. This function checks a session variable which defines whether a user is authenticated or not. If the user is authenticated, no value is returned. If the user is not authenticated, the page is redirected to `login.php`. The default action for `login.php` is to call the `display_login_page` function, which shows the user a form in which to enter a username and password. Once this form is submitted, the `redirect` function is called. This function first does basic checking that the user entered text in for both the username and password, and then validates this information against the `experimenter` table in the database. Upon failure, the user is redirected back to the `display_login_page` function. Upon success, several session variables are set to facilitate global site usage: experimenter id (eid), username, first name (firstname), last name (lastname), administrative status (admin), and authentication status (authenticated). Finally, the user is then redirected to the main `stats.php` page.
b. Admin / User Central Page

Usage Instructions

A user will have different access to database information depending on the administrative status of that user. If a user is listed as an “Admin” in the database, then superuser access is granted, and all site functions are available, as well as all experimenter and subject information. If a user is not listed as an “Admin,” then that user has limited access to subject, experimenter and experiment information. On the main Admin / User Central page, all users will see three primary sections: User Details, Quick Launch Functions, and links to detailed information from individual database tables. Each of these sections is explained in greater detail in the following segments.

Development Notes
The default action for stats.php is to first check the user’s authentication status, then check the user’s administrative status, and finally to display the appropriate initial page. The administrative page is called via the initial function and the regular user page is called via the initial_user function. It should be noted that the necessary HTML header and footer tags are contained within the html_header and html_footer functions respectively. These functions are called to start and end a page, in order to increase the modularity of the other site functions. Both the initial and initial_user functions take care of making several database calls in order to display the appropriate information outlined in the following segments to the user. Both functions also call the user_details function to display detailed information about the current user.

c. User Details Information

Usage Instructions

This section contains pertinent information about the logged in user’s account, including the Experimenter ID, and how many experiments, subjects and protocols are associated with the specified user account.

Development Notes

User details are handled by the user_details function. All of the necessary information is pulled from both session variables and the experiments, subjects, and protocols tables in the database.
d. Experiment Details Page

**Usage Instructions**

This section contains all information about experimental results. On the initial page, the following information is displayed: Experiment ID, EID, Subject Number, Date, Condition ID, Data ID, Unit Number, a link to the Previous and Next related experiments, a link to the raw data files, and a link to view the results as a graph. The Subject Number field is also a link that will display the “Experimental Results for Specified Subject” page, described in more detail in section 5.a. If the specified experiment is a part of a series of experiments, then links to the previous and next experiments will be shown. Otherwise, this field shows “N/A.” The data file links allow the user to download both the header file and the data file as text files, which is
the original unedited storage format of the acquired experimental data. The graph link provides a graph of the results contained in the data file, and opens in a new window.

There is a “Select” field as well, which provides the user with the capability of deselecting certain experimental results from the results table. This paring down of result records is accomplished via a row of buttons at the top of the table, each of which is further described in the following sections.

**Development Notes**

The Experimental Details page calls action 11, SHOW_DETAILED_EXPERIMENTS, in stats.php. In turn, that case first calls the
check_login function and then calls the experimentdetails function. That function handles displaying the appropriate information. It is capable of displaying all experimental results, a single requested result, or a subset of all results as defined by the “Select” choices of the user.

### i. View Selected Results Page

**Usage Instructions**

The last column of the results table provides a checkbox in which each individual result can be “selected” or not. The default is that all results are included. The experimenter can choose to omit results from the display by un-checking the “select” box for those results that are not pertinent to the current viewing session. After making all necessary selection choices, the View Selected Results button refreshes the results to remove omitted entries.

**Development Notes**

Choosing to view only selected results actually calls the same function as viewing all experimental results, experimentdetails, however it passes the “consolidated” variable as “yes” rather than the default of “no.” When the results are being displayed, a check is performed to determine whether each checkbox was deselected. If the check finds that both the checkbox was deselected and “consolidated” is “yes,” then that particular result is skipped during the display loop.
ii. View Selected Results as Thumbnails Page

Usage Instructions

Once appropriate selection choices have been made as described in the previous section, the View Selected Results as Thumbnails button displays all the chosen results as a series of thumbnail graphs of the experimental results. This allows the experimenter to see all of the experiment graphs without having to click “View” for each individual result. Once on the thumbnails page, the experimenter can view a larger version of each graph simply by clicking on it.

Development Notes
Similar to the View Selected Results algorithm, a check is performed to see whether the “select” checkbox was un-checked by the user, and if so, that result is omitted from the results display.

iii. View All Results Page

Usage Instructions

The View All Results button is essentially a “reset” button for the experimental results. If the user has made any number of selection choices that resulted in results being omitted from the displayed results table, this option reverts back to displaying all the results in table form.

Development Notes

This choice simply calls the experimentdetails function with default variables, so that all selection choices are ignored and all results are displayed.

e. Quick Launch Functions

Usage Instructions

Currently there is only one Quick Launch function available which enables an experimenter to view all related experimental results for a specified subject number. Subject numbers are not required to be unique values in the database across different experimenters. Thus, it is possible for two separate experimenters to have identical subject numbers. Regular users will only see results pertaining to their own experiments when using this function. Admin users, however, will see all results for
all experimenters who have utilized that subject number. They can then sort that information by EID, which is the experimenter ID.

**Development Notes**

The Quick Launch function calls action 16, SUBJECT_RESULTS, in *stats.php*. In turn, that case first calls the `check_login` function and then calls the `subjectresults` function. That function handles displaying the appropriate information for regular and admin users. Regular users see results where the specified `subject_number` and the current `eid` match information in the database. Admin users see all results without the limitation of an `eid` check.

i. **Experimental Results for Specified Subject Page**

![Figure 6](image_url)

**Usage Instructions**

The information displayed on this page is very similar to that shown in the Experiment Details page, however the results are limited to those experiments that match the specified `subject_number`. Additional display choices are shown at the
top: View Selected Results, View Selected as Thumbnails, and View All Results. Each of these options is explained in further detail below.

1. View Selected Results Page

*Usage Instructions*

The last column of the results table provides a checkbox in which each individual result can be “selected” or not. The default is that all results are included. The experimenter can choose to omit results from the display by unchecking the “select” box for those results that are not pertinent to the current viewing session. After making all necessary selection choices, the View Selected Results button refreshes the results to remove omitted entries.

*Development Notes*

Choosing to view only selected results actually calls the same function as viewing all results for a subject, *subjectresults*, however it passes the “consolidated” variable as “yes” rather than the default of “no.” When the results are being displayed, a check is performed to determine whether each checkbox was deselected. If the check finds that both the checkbox was deselected and “consolidated” is “yes,” then that particular result is skipped during the display loop.

2. View Selected Results as Thumbnails Page
**Usage Instructions**

Once appropriate selection choices have been made as described in the previous section, the View Selected Results as Thumbnails button displays all the chosen results as a series of thumbnail graphs of the experimental results. This allows the experimenter to see all of the experiment graphs without having to click “View” for each individual result. Once on the thumbnails page, the experimenter can view a larger version of each graph simply by clicking on it.

**Development Notes**

Similar to the View Selected Results algorithm, a check is performed to see whether the “select” checkbox was un-checked by the user, and if so, that result is omitted from the results display.

3. **View All Results Page**

**Usage Instructions**

The View All Results button is essentially a “reset” button for the experimental results. If the user has made any number of selection choices that resulted in results being omitted from the displayed results table, this option reverts back to displaying all the results in table form.

**Development Notes**
This choice simply calls the *subjectresults* function with default variables, so that all selection choices are ignored and all results are displayed.

**f. Experimenter Details Page**

This section allows a regular user to view his/her registration details, and an admin user to view all register users’ details, including name, username and contact information. A regular user may click the “Edit” action button to change the registration information, or the “Delete” action button to terminate the account. An admin user may also perform these same actions but for every registered user. This also provides a method for password reset as explained below in section 5.b.
Development Notes

The expdetails function in functions.php is responsible for displaying experimenter details. The function first determines whether the logged in user is an admin or not, which in turn determines whether all users’ information or only the logged in user’s information is pulled from the database. The information is then displayed in tabular form.

i. Add New Experimenter Page

![Figure 8](image)

Usage Instructions

This section allows for registration of a new user. Both regular users and admin users have rights to add a new user. When a new user is added, it is assumed that he/she will not be an admin. However, after the registration is complete, an admin may go back and set the new user to be an admin as well.
Development Notes

When “Add a New Experimenter” is clicked, first the `check_login` function is called, and then the `solicit_expinfo` function is called. The latter function displays the necessary HTML form elements. Upon submission of the form, `check_login` is again called to ensure the current session has not expired, and then the `add_expinfo` function is called. This function simply collects all the POST variables and inserts the information into the `experimenter` table in the database.

ii. Edit Record Function

![Edit Information](image)

Figure 9

Usage Instructions

When the “Edit” action is chosen, the selected user’s information is loaded into editable fields on a new page. The password is included as well, however, it is hidden such that an admin user may not view the user’s current password, only
change it for the purposes of a password reset. An admin user can set a regular user to be an admin, and also revoke admin status if necessary.

*Development Notes*

The “Edit” action calls the UPDATE_ENTRY case in *update.php*, sending it the type of record to be updated, which in this case is “experimenter,” and the eid of the user for the selected record. The UPDATE_ENTRY case then checks the user’s login status via *check_login* and then calls *update_entry*, passing the function the type of record as a variable. The *update_entry* function pulls the eid from the POST variable itself, and then proceeds to display a similar form to the “Add a New Experimenter” page. The primary difference is that the fields are all editable and filled in with the selected user’s information, pulled from the database. If the logged in user is an admin, then an additional choice of changing the selected user’s admin status will be given. Otherwise, the Administrator status is shown as “No.” When the form is submitted, the *submit_update* function is called, with the type of record again being passed as a variable. This function collects all the POST variables and inserts the information into the appropriate table in the database, determined by the *record* variable. In both the *update_entry* and *submit_update* functions, case statements handle differences in how the form should be displayed and how information is sent to the database, depending on the type of record in question.

### iii. Delete Record Function
Usage Instructions

The “Delete” action takes the user to a confirmation page, to ensure that termination of the user account is the desired action. The user is warned that the action cannot be reversed. Users should be very careful with this action, as should admins, because removing a user’s account also deletes all information associated with that user, including related subjects and experiments.

Development Notes

The “Edit” action calls the CONFIRM_DELETE case in update.php, sending it the type of record to be deleted, which in this case is “experimenter,” and the eid of the user for the selected record. First, check_login is called and then confirm_delete which displays warning text to the user that the action cannot be undone, and forcing them to confirm the delete action. If the user confirms the deletion, then delete_record is called. This function pulls the eid of the selected user from the POST variable. A series of database calls then follows. All information related to the experimenter must be deleted. The database is configured with cascading deletes where possible, which means that deleting the
A user record from the *experimenter* table will automatically remove all related *subject*, *conditions*, *data*, and *experiments* data. However, the *values*, *startAS*, and *start_stim* entries are handled manually via SQL statements.

### g. Subject Details Page

<table>
<thead>
<tr>
<th>EID</th>
<th>Subject #</th>
<th>Gender</th>
<th>Age</th>
<th>Race</th>
<th>Handedness</th>
<th>Genetic BG</th>
<th>Neuro. Assess.</th>
<th>Comments</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>02633-0455</td>
<td>M</td>
<td>25</td>
<td>Asian</td>
<td>R</td>
<td>N/A</td>
<td>Normal</td>
<td>N/A</td>
<td><img src="Yes" alt="Icon" /> <img src="Yes" alt="Icon" /></td>
</tr>
<tr>
<td>6</td>
<td>3780-0876</td>
<td>M</td>
<td>25</td>
<td>Asian</td>
<td>R</td>
<td>Chronic Pain</td>
<td>N/A</td>
<td><img src="Yes" alt="Icon" /> <img src="Yes" alt="Icon" /></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>4567-0228</td>
<td>M</td>
<td>25</td>
<td>Asian</td>
<td>R</td>
<td>Chronic Pain</td>
<td>N/A</td>
<td><img src="Yes" alt="Icon" /> <img src="Yes" alt="Icon" /></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>02633-0455</td>
<td>M</td>
<td>25</td>
<td>Asian</td>
<td>R</td>
<td>Chronic Pain</td>
<td>N/A</td>
<td><img src="Yes" alt="Icon" /> <img src="Yes" alt="Icon" /></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3446-5610</td>
<td>M</td>
<td>25</td>
<td>Asian</td>
<td>R</td>
<td>Chronic Pain</td>
<td>N/A</td>
<td><img src="Yes" alt="Icon" /> <img src="Yes" alt="Icon" /></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>5906-2265</td>
<td>M</td>
<td>25</td>
<td>Asian</td>
<td>R</td>
<td>Chronic Pain</td>
<td>N/A</td>
<td><img src="Yes" alt="Icon" /> <img src="Yes" alt="Icon" /></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4458-9863</td>
<td>M</td>
<td>25</td>
<td>Asian</td>
<td>R</td>
<td>Chronic Pain</td>
<td>N/A</td>
<td><img src="Yes" alt="Icon" /> <img src="Yes" alt="Icon" /></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4567-0228</td>
<td>M</td>
<td>25</td>
<td>Asian</td>
<td>R</td>
<td>Chronic Pain</td>
<td>N/A</td>
<td><img src="Yes" alt="Icon" /> <img src="Yes" alt="Icon" /></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>5674-7621</td>
<td>M</td>
<td>25</td>
<td>Asian</td>
<td>R</td>
<td>Chronic Pain</td>
<td>N/A</td>
<td><img src="Yes" alt="Icon" /> <img src="Yes" alt="Icon" /></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 11**

**Usage Instructions**

The “Subject Details” page displays all information about the experimental subjects. An admin user will see all registered subjects, regardless of the associated experimenter. Regular users will only be able to see subjects that they have registered and are thus associated with their account. The following information is displayed: EID, Subject Number, Gender, Age, Race, Handedness, Genetic Background, Neurological Assessment, and Comments. “Edit” and “Delete” actions
are also an option for each record, and there is an “Add New Subject” button at the bottom. The EID field is shown because the “Subject Number” value is not required to be unique in the database. Thus, it is helpful for admin users to be able to sort the information by experimenter. The “Subject Number” field is also a link that will display the “Experimental Results for Specified Subject” page, described in more detail in section 5.a.

Development Notes

The “Subject Details” page utilizes the SHOW_DETAILED_SUBJECTS case in stats.php. That case first calls the check_login function, and then calls the subdetails function. Subdetails first checks the SESSION variable “admin” to determine whether the logged in user is an admin or not. If so, all the subject information is pulled from the database. If not, then only subjects which are associated with the logged-in user’s eid are pulled. The resulting information is then displayed in tabular form.

i. Add New Subject Page
**Usage Instructions**

The “Add Subject” page allows a user to enter all necessary information about a new user and register them in the database. Each user is required to be associated with an experimenter. When regular users visit this page, any subject that is registered is automatically associated with the logged-in experimenter. When an admin user registers a subject, he/she is provided with an additional field in order to specify the associated experimenter. Since this particular web function will be the primary function used by experimenters, roll-over help text has been implemented for each field. If the user would like help in determining exactly what information is supposed to go into each field, rolling the mouse over the field title will yield a pop-up box containing descriptive help text.

**Development Notes**
When “Add a New Subject” is clicked, the SOLICIT_SUBINFO case in stats.php handles first calling the check_login function, and then calling the solicit_subinfo function. The latter function displays the necessary HTML form elements. The pop-up help text is display via JavaScript onmouseover events. Upon submission of the form, check_login is again called to ensure the current session has not expired, and then the add_subinfo function is called. This function simply collects all the POST variables and inserts the information into the subject table in the database. It also performs some basic checking to ensure that the experimenter did not leave any fields blank, thus ensuring that all required information has been entered for the new subject.

ii. **Edit Record Function**

![EDIT INFORMATION](image)

**Figure 13**
**Usage Instructions**

When the “Edit” action is chosen, the selected subject’s information is loaded into editable fields on a new page, in a format very similar to the “Add Subject” page. Regular users will not be able to change the associated experimenter, but admin users will see an additional “EID” field which will allow them to change the experimenter association for a subject. This field, like the others, is automatically filled in with the correct information, though it can be changed.

**Development Notes**

The “Edit” action calls the UPDATE_ENTRY case in update.php, sending it the type of record to be updated, which in this case is “subject,” and the sid of the subject for the selected record, which is the unique database identified for a subject. The UPDATE_ENTRY case then checks the user’s login status via check_login and then calls update_entry, passing the function the type of record as a variable. The update_entry function pulls the sid from the POST variable itself, and then proceeds to display a similar form to the “Add Subject” page. The primary difference is that the fields are all editable and filled in with the selected subject’s information, pulled from the database. If the logged in user is an admin, then an additional choice of changing the selected subject’s experimenter association will be given. When the form is submitted, the submit_update function is called, with the type of record again being passed as a variable. This function collects all the POST variables and inserts the information into the appropriate table in the database, determined by the record variable. In both the
update_entry and submit_update functions, case statements handle differences in how the form should be displayed and how information is sent to the database, depending on the type of record in question.

### iii. Delete Record Function

![Delete Record](Image)

**Usage Instructions**

The “Delete” action takes the user to a confirmation page, to ensure that deletion of the subject record is the desired action. The user is warned that the action cannot be reversed. Users should be very careful with this action, as should admins, because removing a subject also deletes all information associated with that subject, including experiments in which that subject has participated.

**Development Notes**

The “Edit” action calls the CONFIRM_DELETE case in *update.php*, sending it the type of record to be deleted, which in this case is “subject,” and the sid of the subject for the selected record. First, *check_login* is called and then *confirm_delete* which displays warning text to the user that the action cannot be undone, and forcing them to confirm the delete action. If the user confirms the
deletion, then \textit{delete\_record} is called. This function pulls the sid of the selected user from the POST variable. The database is configured with cascading deletes where possible, which means that deleting the subject record from the \textit{subject} table will automatically remove all related \textit{experiments} data.

h. Type Details Page

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{cortical_database.png}
\caption{Type Details Page}
\end{figure}

\textbf{Usage Instructions}

The “Type Details” page displays all information about the defined experiment types. The following information is displayed: Type ID, Name, and Description. “Edit” and “Delete” actions are also an option for each record, and there is an “Add New Type” button at the bottom. Type definitions describe the purpose of an experiment; the high-level goal of the specific information collection. Type definitions are useful for classifying experiments, and in future applications, for analysis of data.

\textit{Development Notes}
Clicking “Show Type Details” on the main page calls the SHOW_DETAILED_TYPE case in *stats.php*. The user login status is checked first, and then the function *typedetails* in *functions.php* is called. This function checks the *admin* session variable to determine whether the user is an admin. If so, then the “Edit” and “Delete” actions are displayed and available. Otherwise these actions are not available. All of the protocol information is then pulled from the *type* table in the database, and displayed in tabular form.

i. **Add New Type Page**

![ADD TYPE](image)

**Usage Instructions**

Any user, regular or admin, may add a new type. A type definition consists of a Name and Description. Once added to the database, types are not unique to each experimenter, but rather are shared across all experimenters.

**Development Notes**

The “Add New Type” button calls the SOLICIT_TYPEINFO case in *stats.php* which checks the user login status and then calls *solicit_typeinfo* in
functions.php. This function simply displays a form to the user, allowing a Name and Description to be entered. Upon submission of this form, the add_typeinfo function is called. The POST variables are collected and checked to make sure neither is empty, and then the information is inserted into the type table in the database.

ii. Edit Record Function

![Cortical Database](image)

Figure 17

**Usage Instructions**

Only admin users may edit types. This is to prevent accidental changes to type definitions, which are shared across all experimenters and experiments. Due to the fact that multiple experimenters and experiments may rely on specific type definitions, then changes to that information may have unintended consequences. Admin users should be very careful when editing entries.

**Development Notes**
The “Edit” action calls the UPDATE_ENTRY case in update.php, sending it the type of record to be updated, which in this case is “type,” and the typeid of the selected record, which is the unique database identified for a type. The UPDATE_ENTRY case then checks the user’s login status via check_login and then calls update_entry, passing the function the type of record as a variable. The update_entry function pulls the typeid from the POST variable itself, and then proceeds to display a similar form to the “Add New Type” page. The primary difference is that the fields are all editable and filled in with the selected type record information, pulled from the database. When the form is submitted, the submit_update function is called, with the type of record again being passed as a variable. This function collects all the POST variables and inserts the information into the appropriate table in the database, determined by the record variable. In both the update_entry and submit_update functions, case statements handle differences in how the form should be displayed and how information is sent to the database, depending on the type of record in question.

iii. Delete Record Function

Figure 18

Usage Instructions
Only admin users may delete types for the same reasons mentioned in Section 8.b. Admin users should be very careful when deleting entries.

**Development Notes**

The “Delete” action calls the CONFIRM_DELETE case in *update.php*, sending it the type of record to be deleted, which in this case is “type,” and the typeid of the selected record. First, *check_login* is called and then *confirm_delete* which displays warning text to the user that the action cannot be undone, and forcing them to confirm the delete action. If the user confirms the deletion, then *delete_record* is called. This function pulls the typeid of the selected record from the POST variable. Then a SQL DELETE statement is executed to remove the record from the *type* table in the database.

**i. Protocol Details Page**

![Protocol Details Page](image)

**Usage Instructions**
The “Protocol Details” page displays all information about the defined experiment protocols. The following information is displayed: Protocol ID, Type ID, Metric, and Description. “Edit” and “Delete” actions are also an option for each record, and there is an “Add New Type” button at the bottom. Protocol definitions are used to keep track of the measurement metric of the experiment; i.e. exactly what is being measured. As with a type definition, protocol definitions will be useful in future analysis applications, because it will allow a user to view all experiments that looked at a particular metric.

Development Notes

Clicking “Show Protocol Details” on the main page calls the SHOW_DETAILED_PROTOCOL case in stats.php. The user login status is checked first, and then the function protocoldetails in functions.php is called. This function checks the admin session variable to determine whether the user is an admin. If so, then the “Edit” and “Delete” actions are displayed and available. Otherwise these actions are not available. All of the protocol information is then pulled from the protocol table in the database, and displayed in tabular form.

i. Add New Protocol Page
Usage Instructions

Any user, regular or admin, may add a new protocol. A protocol definition consists of an associated Type ID, Metric and Description. The Type ID is a unique identifier in the database that links to a specific type record. On the “Add Protocol” page, these choices are displayed to the user in a drop-down box. The Metric choice is also a drop-down box, however these are simply pre-defined enumerated values which are a part of the database. Once added to the database, protocols are not unique to each experimenter, but rather are shared across all experimenters.

Development Notes

The “Add New Protocol” button calls the SOLICIT_PROTOCOLINFO case in stats.php which checks the user login status and then calls solicit_protocolinfo in functions.php. This function simply displays a form to the user, allowing a TypeID and Metric to be chosen, and a Description to be entered. The TypeID drop-down box is dynamically populated from information in the typeID table in
the database. Upon submission of this form, the add_protocolinfo function is called. The POST variables are collected and checked to make sure neither is empty, and then the information is inserted into the protocol table in the database.

ii. **Edit Record Function**

![Cortical Database](image_url)

**Usage Instructions**

Only admin users may edit protocols. This is to prevent accidental changes to protocol definitions, which are shared across all experimenters and experiments. Due to the fact that multiple experimenters and experiments may rely on specific protocols, then changes to that information may have unintended consequences. Admin users should be very careful when editing entries.

**Development Notes**

The “Edit” action calls the UPDATE_ENTRY case in update.php, sending it the type of record to be updated, which in this case is “protocol,” and the
protocolid of the selected record, which is the unique database identified for a protocol. The UPDATE_ENTRY case then checks the user’s login status via check_login and then calls update_entry, passing the function the type of record as a variable. The update_entry function pulls the protocolid from the POST variable itself, and then proceeds to display a similar form to the “Add New Protocol” page. The primary difference is that the fields are all editable and filled in with the selected protocol record information, pulled from the database. When the form is submitted, the submit_update function is called, with the type of record again being passed as a variable. This function collects all the POST variables and inserts the information into the appropriate table in the database, determined by the record variable. In both the update_entry and submit_update functions, case statements handle differences in how the form should be displayed and how information is sent to the database, depending on the type of record in question.

iii. Delete Record Function

Figure 22

Usage Instructions
Only admin users may delete protocols for the same reasons mentioned in Section 9.b. Admin users should be very careful when deleting entries.

*Development Notes*

The “Delete” action calls the CONFIRM_DELETE case in `update.php`, sending it the type of record to be deleted, which in this case is “protocol,” and the `protocolid` of the selected record. First, `check_login` is called and then `confirm_delete` which displays warning text to the user that the action cannot be undone, and forcing them to confirm the delete action. If the user confirms the deletion, then `delete_record` is called. This function pulls the `protocolid` of the selected record from the POST variable. Then a SQL DELETE statement is executed to remove the record from the `protocol` table in the database.

**j. Query Builder Page**

![Query Builder](image)

*Usage Instructions*

The purpose of the Query Builder is to provide admin users access to all information in the database, regardless of whether a specific set of queries has been
programmed for that access. An admin user is able to build his/her own SQL query which provides flexibility to access and analyze data in new ways without requiring editing of the front-end PHP code. When the page first loads, only a table value can be chosen. Once that is selected, the user may then choose select and where values. Optionally, the user may also add a second set of conditions to the query via the last drop-down box. This box takes the form of AND, OR, UNLESS, or NOT choices, depending on the relationship of the second set of conditions to the first. This choice then allows the addition of a second set of SQL statement choices in the same vein as the first set of conditions. The user may then click “Build Query” to execute the created SQL statement, the results of which will be displayed in tabular form if possible. Note that it is possible to create an illogical statement that is unable to be executed by the database. If that is the case, the user will be notified as such.

Development Notes

The Query Builder code is all contained within querybuilder.php. If no action is specified with querybuilder.php is loaded, the SHOW_QUERY_FORM case is the default. This case first checks the user login via check_login and then calls the query_form2 function. This function displays the query building form to the user. Initially, only the table can be chosen. When that event occurs, the querybuilder.php page is called again, this time with the table POST variable non-empty. The user may then choose a SELECT value, and a WHERE value. All of these fields are dynamically generated from information in the database, such that any changes to tables in the database are immediately reflected in these drop-down choices.
Choosing a WHERE clause again causes the `querybuilder.php` page to be reloaded, this time with the `table`, `select` and `where` POST variables non-empty. Finally, the user is able to choose a second condition to attach to the query, in the form of an AND, OR, UNLESS, or NOT. This choice then causes a reload, and a second set of SQL statement choices can be made in the same vein as the first set of conditions. Once the user clicks “Build Query”, the `build_query` function is executed. This simply executes the concatenated set of variables. If the function is illogical and does not execute properly, an error message is displayed.

k. **Account Information Page**  
This section is nothing more than an easier way to view the aforementioned Sections 3, 4, and 5 in a single-page view. In this current iteration, there is no additional information displayed on this page that is not available in those Sections. Please refer back to Sections 3, 4, and 5 for usage and development information pertaining to this page.

l. **About Us Page**  
i. **History of Cortical Metrics Page**  
   
   This page will be used to provide more detailed information about the company running the database.

ii. **Contact Cortical Metrics Page**
Usage Instructions

This page provides a basic contact form for users who are either encountering trouble using the site, or have questions about registration or general information concerning the site. A user may enter his/her name, email, and questions/comments. Clicking “Send” emails this information to the site administration team.

Development Notes

The default case is CONTACT_FORM, which calls the contactform function. Note that this is one of the few times that the user’s login status is not verified, because the contact form is available to everyone, included unregistered users. The contactform function displays a simple form which calls contactsubmit upon submission. This function does basic variable checking to ensure that the user has entered valid information, including a valid email, into the form fields. The email is then created, containing additional information about the submitting user.
including the IP address and browser type. The email is then sent using the PHP 
mail function, and the user is shown a confirmation message.
V. Impact and Future Direction

The current testing method, apparatus, and protocols are targeted at the detection of a systemic cerebral cortical alteration. Thus, there are numerous areas of clinical research and basic science and medicine that will potentially be impacted:

1. **Neurodevelopmental Disorders: Autism**
   
   Cortical information processing is significantly impaired in autistic patients. Fast, non-invasive diagnosis of autism would be a boon to the health care industry.

2. **Aging / Alzheimer’s Disease**
   
   The testing method has the potential to aid in the early diagnosis of Alzheimer’s.

3. **Chronic Pain**
   
   While chronic pain is highly prevalent in the geriatric population, there are currently few tools and procedures to help in the diagnosis of the variety of chronic pain conditions.

4. **Sleep Deprivation & Fatigue**
   
   It is anticipated that additional research, supported by the current project, will lead to novel pharmaceuticals that would benefit not only a war fighter’s ability to function for an extended period of time, but prove beneficial to other professionals that operate on reduced sleep.

5. **Acute Head Trauma**
Both acute head trauma and traumatic brain injury (TBI) affect cortical information processing, and thus the testing method can aid in the decision of whether to return an athlete to competition after mild head injury, or allow military commanders to easily determine the extent of TBI for a soldier.

6. Pharmacological Effects

The testing method provides additional quantitative metrics on the effects of drugs on the central nervous system.

7. Cerebral Cortical Health

The establishment of a “normal” baseline values from a significantly large database of control subjects could provide a method for evaluating the general cortical health of a patient. It is anticipated that cerebral cortical health could be assessed similar to the way that blood pressure is used as a metric of the health of the heart and circulatory system.

The general usefulness of the testing method and apparatus is in diagnosis, which requires a comparison of test data against baseline data. The centralized database plays the largest role in enabling both comparison and data mining analysis of experimental data. In the current iteration, the database provides the framework for data collection, and provides methods for sharing and classifying data. However, the true power of the database has yet to be tapped to provide easy analytical evaluation of submitted research data. The future direction of the database is two-fold: 1) extensibility and 2) cross-correlation functionality.
The extensibility of the current database structure allows it to remain adaptable to changing experimental focus. There are many ways in which the database structure could be extended to support additional information gathering and/or analysis:

1. **Pharmacological Information**

   Currently, the pharmacological state of a subject is contained within categorization of “Substance Abuse”, and thus does not allow for changes across experiments without the creation of a new subject. This particular category of information, given the future potential impact experimental data can have on pharmacological effects, may be better suited for its own category separated from the subject information and associated with the experiment instead. Additionally, this would allow for a broad range of pharmacological classifications, and thus for analysis of drug interactions and effects on experimental results.

2. **Genotype**

   The inclusion of tracking certain genetic markers could be highly relevant in determining the effects those markers have on the performance of the subject in the testing method, and thus clarifying their role in cortical information processing.

3. **Family History**

   This subset of information would be geared toward the future clinical applications of the testing method. Knowing a patient’s family history can aid in the analysis of correlated performance metrics, and potentially serve as an indicator of predicted test results.

4. **Baseline Tracking**
The database could be modified to support moving average tracking of baseline performance results for all subjects classified as “normal” in neurological assessment. This could provide researchers and clinical users with instant feedback of subject comparison with a large and growing set of baseline values, for future evaluations of cerebral cortical health.

As the future direction of the research is more clearly refined, changes to the underlying database structure will not only be possible, but will enable new rounds of research interpretation and analysis.

Cross-correlation of disparate data sources is where the true power of a relational database lies. A relational database structure allows a research to view and compare experimental data against control data, and view specific changes in results as a function of alternate variables without requiring an entirely new set of protocols and experiments. Due to the modular nature of the database, individual data components can be classified and, in the future, separated and re-combined in new ways. As in the aforementioned example, one future analysis might compare result sets as a function of pharmacological interactions. As the database grows larger and researchers begin developing and adding new protocols to the system, new potential analytical concepts will arise that may not have easily been foreseen. Therein lies the desirability of a centralized, relational research database.
APPENDIX A

Database Structure

- CONDITIONS
  - conditionID
  - protocolID
  - start_Astim1ID
  - start_Astim2ID
  - start_Astim3ID
  - start_Astim4ID
  - start_stim1ID
  - start_stim2ID
  - start_stim3ID
  - start_stim4ID
  - lead_lag_time
  - adapter_test_interval
  - num_trials
  - bias
  - step_size
  - max_step_size
  - min_step_size
  - trial_to_half_step_size
  - max_value
  - min_value

- DATA
  - dataID
  - start_valID
  - end_valID

- EXPERIMENTER
  - eid
  - eUsername
  - ePassword
  - eFirstName
  - eLastName
  - eStreet1
  - eStreet2
  - eCity
  - eState
  - eEmail
  - eAdmin

- EXPERIMENTS
- expID
- eid
- sid
- conditionID
- dataID
- unit_id
- prev_expID
- next_expID
- date
- dataFile1
- dataFile1name
- dataFile1size
- dataFile2
- dataFile2name
- dataFile2size
- graph
- omit
- timestamp

- PROTOCOL
  - protocolID
  - typeID
  - metric
  - description

- START_STIM
  - startstimID
  - amp
  - freq
  - indent_offset
  - indent_stim_delay
  - phase
  - stim_duration
  - intertrial_interval
  - interstim_interval
  - start_temp
  - probe_tip_diam
  - probe_tip_shape

- SUBJECT
  - sid
  - eid
  - subject_number
  - gender
  - age
  - race
  - genetic_BG
  - neuro_assess
  - handedness
• comments

• TYPE
  o typeID
  o name
  o description

• VALS
  o valID
  o timestamp
  o val
  o stim_order
  o response_order
  o temp
  o response_time
APPENDIX B

Relational Hierarchy

![Diagram of Relational Hierarchy]

Figure 25
APPENDIX C

SQL Database Creation Code

--
-- Create the database
--

-- CREATE DATABASE corticalmetrics
--

USE corticalmetrics;
SELECT DATABASE();

CREATE TABLE data (
    dataID BIGINT unsigned NOT NULL AUTO_INCREMENT,
    PRIMARY KEY (dataID),
    start_valID BIGINT unsigned NOT NULL,
    end_valID BIGINT unsigned NOT NULL
) ENGINE=InnoDB;

CREATE TABLE vals (
    valID BIGINT unsigned NOT NULL AUTO_INCREMENT,
    PRIMARY KEY (valID),
    timestamp TIME NOT NULL,
    val INT unsigned NOT NULL,
    stim_order TINYINT unsigned,
    response_order TINYINT unsigned,
    temp INT(3) NULL
) ENGINE=InnoDB;

CREATE TABLE experimenter (
    eid INT unsigned NOT NULL AUTO_INCREMENT,
    PRIMARY KEY (eid),
eUsername VARCHAR(50) NOT NULL UNIQUE,
ePassword VARCHAR(50) NOT NULL,
eFirstName VARCHAR(50) NOT NULL,
eLastName VARCHAR(50) NOT NULL,
eStreet1 VARCHAR(50) NULL,
eStreet2 VARCHAR(50) NULL,
CREATE TABLE subject (  
sid INT unsigned NOT NULL AUTO_INCREMENT,  
eid INT unsigned NOT NULL,  
PRIMARY KEY (sid, eid),  
INDEX (eid),  
subject_number TEXT NOT NULL,  
gender ENUM('F','M') NOT NULL default 'M',  
birthdate DATE NOT NULL,  
race ENUM('Caucasian','Black','Asian','Hispanic','Native American','Other') NOT NULL default 'Other',  
genetic_BG LONGTEXT NULL,  
neuro_assess ENUM('Aging','Alzheimers','Autistic','Chronic Pain','Diabetes','Menopause','Normal','Schizophrenic','Substance Abuse','TBI','Other') NOT NULL default 'Normal',  
handedness ENUM('L','R') NOT NULL default 'R',  
comments LONGTEXT NULL,  
FOREIGN KEY (eid) REFERENCES experimenter (eid)  
ON DELETE CASCADE  
) ENGINE=InnoDB;

CREATE TABLE type (  
typeID INT unsigned NOT NULL AUTO_INCREMENT,  
PRIMARY KEY (typeID),  
name TEXT NOT NULL,  
description LONGTEXT NOT NULL  
) ENGINE=InnoDB;

CREATE TABLE protocol (  
protocolID INT unsigned NOT NULL AUTO_INCREMENT,  
typeID INT unsigned NOT NULL,  
PRIMARY KEY (protocolID, typeID),  
INDEX (typeID),  
metric ENUM('Distance','Amplitude','Frequency') NOT NULL,  
description LONGTEXT NULL,  
FOREIGN KEY (typeID) REFERENCES type (typeID)  
ON DELETE CASCADE  
) ENGINE=InnoDB;

CREATE TABLE start_stim (  
startstimID BIGINT unsigned NOT NULL AUTO_INCREMENT,  
PRIMARY KEY (startstimID),  
FOREIGN Key (startstimID) REFERENCES protocol (protocolID)  
) ENGINE=InnoDB;
amp FLOAT NOT NULL,
freq FLOAT NOT NULL,
indent_offset SMALLINT NOT NULL,
indent_stim_delay MEDIUMINT NOT NULL,
phase SMALLINT NOT NULL,
stim_duration INT NOT NULL,
intertrial_interval INT NOT NULL,
interstim_interval INT NOT NULL,
start_temp TINYINT NULL,
probe_tip_diam SMALLINT NOT NULL,
probe_tip_shape ENUM('circular','square','bar')
) ENGINE=InnoDB;

CREATE TABLE conditions (conditionID BIGINT unsigned NOT NULL AUTO_INCREMENT, protocolID INT unsigned NOT NULL, startASID BIGINT unsigned NOT NULL, PRIMARY KEY (conditionID, protocolID, startASID), INDEX (protocolID), INDEX (startASID), start_Astim1ID BIGINT NULL, start_Astim2ID BIGINT NULL, start_Astim3ID BIGINT NULL, start_Astim4ID BIGINT NULL, start_stim1ID BIGINT NULL, start_stim2ID BIGINT NULL, start_stim3ID BIGINT NULL, start_stim4ID BIGINT NULL, lead_lag_time INT NOT NULL, adapter_test_interval INT NOT NULL, num_trials SMALLINT NOT NULL, bias TINYINT NOT NULL, step_size FLOAT NOT NULL, max_step_size FLOAT NOT NULL, min_step_size FLOAT NOT NULL, trial_to_half_step_size SMALLINT NOT NULL, max_value FLOAT NOT NULL, min_value FLOAT NOT NULL, physical_mod ENUM ('alcohol','anesthetic','medication'), FOREIGN KEY (protocolID) REFERENCES protocol (protocolID) ON DELETE CASCADE, FOREIGN KEY (startASID) REFERENCES startAS (startASID) ON DELETE CASCADE ) ENGINE=InnoDB;

CREATE TABLE experiments (
expID INT UNSIGNED NOT NULL AUTO_INCREMENT,
eid INT unsigned NOT NULL,
sid INT unsigned NOT NULL,
conditionID BIGINT unsigned NOT NULL,
dataID BIGINT unsigned NOT NULL,
PRIMARY KEY (expID, eid, sid, conditionID, dataID),
INDEX (eid),
INDEX (sid),
INDEX (conditionID),
INDEX (dataID),
unit_id INT unsigned NOT NULL,
prev_expID INT unsigned NULL,
next_expID INT unsigned NULL,
date DATE NOT NULL,
dataFile1 LONGBLOB,
dataFile1name TINYTEXT NULL,
dataFile1size INT unsigned NULL,
dataFile2 LONGBLOB,
dataFile2name TINYTEXT NULL,
dataFile2size INT unsigned NULL,
graph BLOB,
omit TINYINT(1) unsigned NOT NULL default 0,
FOREIGN KEY (eid) REFERENCES experimenter (eid),
  ON DELETE CASCADE
FOREIGN KEY (sid) REFERENCES subject (sid),
  ON DELETE CASCADE
FOREIGN KEY (conditionID) REFERENCES conditions (conditionID)
  ON DELETE CASCADE,
FOREIGN KEY (dataID) REFERENCES data (dataID)
  ON DELETE CASCADE
) ENGINE=InnoDB;
APPENDIX D

Data Storage Procedure

1. If experimenter table entry exists GOTO step 2. Else, create experimenter table entry:
   - Obtain variables `username`, `password`, `firstname`, `lastname`, `street`, `city`, `state` from data file.
   - INSERT INTO experimenter
     VALUES(NULL,'username','password','firstname','lastname','street','city','state');
   - Store the `eid` value from this entry.

2. If subject table entry exists GOTO step 3. Else, create subject table entry:
   - Obtain variables `subject_number`, `gender`, `age`, `race`, `genetic_BG`, `neuro_assess`, `handedness`, `comments` from data file.
   - Obtain the `eid` from the experimenter table.
   - INSERT INTO subject
     VALUES(NULL,'subject_number','eid','gender','age','race','genetic_BG','neuro_assess','handedness','comments');

3. Create protocol table entry:
   - Obtain variables `type`, `metric`, `description` from data file.
   - Obtain `typeID` from type table using `type` variable from data file.
   - INSERT INTO protocol VALUES(NULL,'typeID','metric','description');
   - Store the `protocolID` value from this entry.

4. Create starting conditions adapting stimulus table entry:
   - Obtain variables `amp`, `freq`, `indent_offset`, `indent_stim_delay`, `phase`, 
     `stim_duration`, `intertrial_interval`, `interstim_interval`, `start_temp`,
     `probe_tip_diam`, `probe_tip_shape` from data file.
   - INSERT INTO startAS
     VALUES(NULL,'amp','freq','indent_offset','indent_stim_delay','phase',
     'stim_duration','intertrial_interval','interstim_interval','start_temp','probe_tip_diam','probe_tip_shape');
   - Store the `startASID` value from this entry.

5. Create starting conditions stimulus table entries, repeat as necessary:
   - Obtain variables `amp`, `freq`, `indent_offset`, `indent_stim_delay`, `phase`, 
     `stim_duration`, `intertrial_interval`, `interstim_interval`, `start_temp`,
     `probe_tip_diam`, `probe_tip_shape` from data file.
   - INSERT INTO start_stim
     VALUES(NULL,'amp','freq','indent_offset','indent_stim_delay','phase',
     'stim_duration','intertrial_interval','interstim_interval','start_temp','probe_tip_diam','probe_tip_shape');
   - Store the `startstim[1..8]ID` value from this entry.

6. Create conditions table entry:
o Obtain variables lead_lag_time, adapter_test_interval, stim_order, response_order, num_trials, bias, step_size, max_step_size, min_step_size, trial_to_half_step_size, max_value, min_value from data file.

o Recall protocolID, startASID, start_stim[1..8]ID from previous steps.

o INSERT INTO conditions
VALUES(NULL, 'protocolID', 'startASID', 'start_stim1ID', 'start_stim2ID', 'start_stim3ID', 'start_stim4ID', 'start_stim5ID', 'start_stim6ID', 'start_stim7ID', 'start_stim8ID', 'lead_lag_time', 'adapter_test_interval', 'stim_order', 'response_order', 'num_trials', 'bias', 'step_size', 'max_step_size', 'min_step_size', 'trial_to_half_step_size', 'max_value', 'min_value');

o Store conditionID value from this entry.

7. Create values table entries:
   o For each entry in the data file, obtain variables timestamp, value, temp.
   o INSERT INTO vals VALUES(NULL, 'timestamp', 'val', 'temp');
   o Store the valID value for the first and last entries.

8. Create data table entry:
   o Recall the two valID values from previous steps.
   o INSERT INTO data VALUES(NULL, 'start_valID', 'end_valID');

9. Create experiments table entry:
   o If this function was called nested, then recall the prev_expid passed to the function.
   o Recall eid, sid, conditionID, dataID from previous steps.
   o Obtain variables unit_id, date from data file.
   o Store the data files themselves as binary files to be passed as variables to the database.
   o INSERT INTO experiments
VALUES(NULL, 'eid', 'sid', 'conditionID', 'unit_id', 'dataID', 'prev_expid', NULL, 'date', 'dataFile1', 'dataFile2');
   o Store the expID value from this entry.
   o UPDATE experiments SET next_expid='expID' WHERE expID='prev_expid';
   o If there is a subsequent experiment in the sequence, repeat this function and pass the current expID as prev_expid.
REFERENCES


