# iHUMAN FINAL PROJECT REPORT
# DATABASE DESIGN

April 16, 2015

**iHuman Youth Society**
&
**Dr. Sanjiv Shrivastava**
sanjiv.shrivastava@yahoo.com

**Database Designers**

**Kara Au**
kwau@ualberta.ca
506-259-3849

**Grady Zielke**
gzielke@ualberta.ca
780-278-1565

**Front-End Interface Designers**

**John Nguyen**
jncustoms@outlook.com

**Alex Adu-Darko**
cassherv3@gmail.com

As of this report, the final example version of the database can be viewed at:
**http://hucodev.srv.ualberta.ca/kwau/iHuman/index.php**
(Subject to removal at any time, should not be used as the official database, when it is removed all user data will be lost.)

Future inquiries about the database should be forwarded to Grady Zielke.

## Overview

Our contribution to the project is composed of a back-end database for the Moving the Mountain (MTM) project to document and securely store the personal progress and points collected by youth. Additionally, we created a front-end prototype HTML interface for the database to demonstrate current progress and what it is capable of doing.

## Original Goal

To have a database and library of functions capable of being compatible with the Construct 2 front-end interface that Alex and John were working on.

## Our Final Progress / What We Accomplished

The project changed drastically halfway through its development. In addition to the database and the library, we included a web page that the user can navigate to and use the database in its current progress.

Below is the final table of what we accomplished, with edits:

| | |
|---|---|
| January 22 | ~~Submit initial project plan.~~ |
| February 5 | ~~Familiarizing ourselves with current MTM documentation and code.~~ |
| February 19 | ~~Designing the database to fit the project's needs.~~ |
| February 28 | ~~Submitting mid-project report.~~ |
| March 12 | ~~Implementing and documenting a working prototype of the database.~~ |
| April 2 | Have a functions library in JavaScript** and ~~a basic front-end HTML interface for testing.~~ |
| April 9 | ~~Troubleshooting and bugfixing complete.~~ |
| April 16 | ~~Final implementation into front-end interface complete & hand-off of final project.~~ |
| Stretch Goals | Bonus Points<br>~~Personalized points system where individual accounts can have unique point values for events if set by administrator.~~ |

** The functions library is programmed in PHP, but has the capability of encoding its returns in JavaScript if required.

Overall, we accomplished most of what we wanted to do and also managed to reach a few of our stretch goals. The only thing that changed from our original plan is that currently the database is not optimized to export directly into a JavaScript environment, but it can be made to do so with only minor modifications. We also completed a much more sophisticated and usable HTML page than we had initially planned on making, and connected it fully with our database. It can work as a suitable stop-gap solution until the final front-end interface is finished. We are leaving the project before the database is fully integrated into the front-end interface created by John and Alex, but Grady will remain in contact to provide support for them as they work towards integrating the database with their interface.

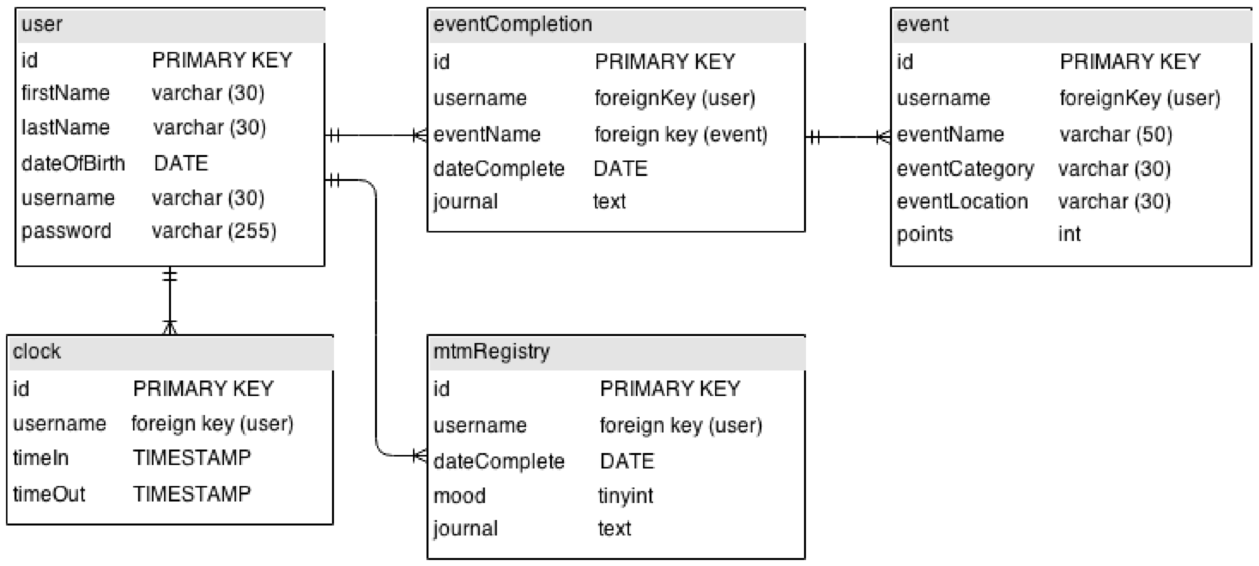
**Technical Specifications**
There are several technologies at work with the database and the HTML front-end we have designed. The database itself is the core of our portion of the project. It is written in MySQL, an open source database which is free to use and has ongoing support from its community. It is common in both academic and industry projects. We chose this technology because of its cost and its reliability. Our database is usable on any server or computer that has MySQL software installed.

There is also a library of functions to allow for easier access to the database. This library is written in PHP. PHP is one of the most commonly used programming languages in web servers and has wide-ranging support for conversions between it and various other languages. Most importantly, PHP is capable of encoding data to be readable by JavaScript, which we were told Construct 2 had access to. PHP is free, but it is not open source. Our library of functions can be run on any server that has a PHP parser. Currently, our library of functions is not optimized to connect with any other technology than the HTML pages included with this project, but Grady is available to help with support and changes to the library this summer if any issues arise.

The final piece of our project is the HTML page that we connected to our database. The page itself can be accessed on the internet, if it is hosted on a web server, or locally, if it is stored locally. It can be opened with any internet browser, with the possible exceptions of old or out-of-date browsers. It has been thoroughly tested in Google Chrome and Firefox. This HTML page is capable of connecting to and manipulating our database, and allows youth to input events and successes and document them with journal entries, as was the intention of the project.

The HTML page was not intended to be the final implementation of the project; rather, it is an interface which should allow both youth and administrators to interact with the database and make use of it while the final implementation is created. Some JavaScript is also used on the log in HTML page, but it is used for cosmetic and usability reasons and does not affect the rest of the project.

**MySQL Database Design**



After much deliberation, this is the final database design that we settled on. Each table has a PRIMARY KEY id, which is a number that identifies each unique entry in each table.

The foreign key denotes times when we must reference values from other tables. For instance, in the clock table, to know who is signing in and out we need to fetch that user's PRIMARY KEY id from the table user.

The term varchar denotes string entries (letters). The number next to it is the max amount of characters the database can store. For instance, the world "username" has 8 characters (u-s-e-r-n-a-m-e).

The term int, or integers, are numbers.

Finally, text is self-explanatory. It can hold a large amount of text, and keeps the journal entry formatting in-tact, so users can insert spaces for paragraphs if they wish.

**Functions Library**

The following is a list of all of the functions used in the library to connect to the database and process information. It contains all of the comments and documentation of what those functions are, how to call them, and what they do. All of this documentation is also included in the actual library file, which can be accessed by opening it in a text editing program like Notepad. Microsoft Word will likely not work.

```
insertNewUser($dbPipeline, $year, $month, $day, $firstname,
$lastname, $username, $password)
```

This function TAKES the database connection, a year of birth, a month of birth, a day of birth, a first name, a last name, a username, and a password and RETURNS nothing. This function updates the database with a new user complete with birth date, first and last name, username, and password.

Example:
```
insertNewUser($databaseConnection, $yearThey'reBorn,
$MonthThey'reBorn, $dayThey'reBorn, $firstName, $lastName,
$username, $password);
```

This function doesn't log you in, it creates a new set of login information. This also checks to see if there is already someone with the selected username before adding that username to the database. It doesn't allow for multiple instances of the same username, but does allow multiples of other features.

**`combineDate($year, $month, $day)`**

This function just takes the information from the insertNewUser() function and puts it in a format the database likes.

**`listAllUsers($dbPipeline)`**
This function TAKES the database connection and RETURNS a full list of all the usernames in the database. This function doesn't update the database with new information.

Example:
```
$usernameArray = listAllUsers();
```

The array that is returned, $resultsArray, will have $resultsArray['0'] is the first username. Each username is connected to their id value.

**`listAllEvents($dbPipeline)`**

This function TAKES the database connection and RETURNS a full list of all the events in the database. This function doesn't update the database with new information.

Example:
```
$EventNameArray = listAllEvents($databaseConnection);
```

The array that is returned, $resultsArray, will have $resultsArray['0'] is the first event name. Each event name is connected to their id value.

**`listCurrentUserEvents($dbPipeline, $userId)`**

This function TAKES the database connection and a user id and RETURNS a list of events specific to that user and the admin events. This function doesn't update the database with new information.

Example:
```
$EventNameArray = listCurrentUserEvents($databaseConnection,
$userId);
```

The array that is returned, $results array, will have $resultsArray['*someNumber*'] is an event name. Each event is tied to its id. There's no guarantee that the array starts at one because this function ONLY returns events created by the user that is logged in. It also returns events created by the admin, which allows the database to have a "default" set of events.

**`wrapInOptionsTags($optionArray)`**

This function just wraps things in option tags. It's only really useful in HTML. If you combine it with the listAllEvents() or listAllUsers() functions, if creates a selectable list within a form.

Example:
```
wrapInOptionsTags(listAllUsers());
```

**`insertNewEvent($dbPipeline, $userId, $eventName, $eventCategory, $eventLocation, $points)`**

This function TAKES the database connection, the point value for a new event, the name of a new event, the category for a new event, the location for a new event, and a user's id and RETURNS nothing. This function updates the database with a new event. You can determine the name of the event, its category, location, point value, and the user who created it.

Example:
```
insertNewEvent($databaseConnection, $pointValueOfEvent,
$nameOfEvent, $categoryOfEvent, $locationOfEvent, $userId);
```

NOTE: If the admin creates a function, it is accessible to all users when using the listCurrentUserEvents() function.

**`insertNewCompleteEvent($dbPipeline, $journal, $user, $event)`**

This function TAKES the database connection, a journal entry, a user id, and an event id and RETURNS nothing. This function updates the database with a new completed event by a user. You can access that information with other functions.

Example:
```
insertNewCompleteEvent($databaseConnection, $journalEntry,
$IdOfPersonWhoCompletedAnEvent, $IdOfEventThePersonCompleted);
```

**`insertMood($dbPipeline,$user,$mood,$journal)`**

This function TAKES the database connection, a user id, a mood, and a journal entry and RETURNS nothing. This function updates the database with a new entry into the MTM registry table.

Example:
```
insertMood($databaseConnection, $userId, $moodOnAScaleOf1To5,
$journalEntry);
```

**`timeIn($dbPipeline, $timeInId)`**

This function TAKES the database connection and a user id and RETURNS nothing. This function updates the database with the time that the person whose id you enter checks in.

Example:
```
timeIn($databaseConnection, $IdOfPersonYouWantToCheckIn);
```

**`timeOut($dbPipeline, $timeOutId)`**

This function TAKES the database connection and a user id and RETURNS nothing. This function updates the database with the time that the person whose id you enter checks out.

Example:
```
timeOut($databaseConnection, $IdOfPersonYouWantToCheckOut);
```

**`getEventHistory($dbPipeline, $getEventHistoryId)`**

This function TAKES the database connection and a user id and RETURNS all of the events that a user has completed as arrays within an array. This function doesn't update the database with new information.

Example:
```
$myArray = getEventHistory($databaseConnection,
$idOfPersonYou'reLookingFor);
```

The array that is returned, $historyFinalArray, will have $historyFinalArray['firstName'] is the user's first name, $historyFinalArray['lastName'] is the user's last name, $historyFinalArray['eventName'] is the name of the event completed, $historyFinalArray['eventCategory'] is the name of the category of that event, $historyFinalArray['eventLocation'] is where the event was completed (Library, pool, etc.) $historyFinalArray['points'] is how many points the event was worth, $historyFinalArray['dateComplete'] is the date that the event was completed, and $historyFinalArray['journal'] is the journal entry corresponding to that event. Each complete event is an array within the array. So, $historyFinalArray[0][eventName] gets the name of the first event, historyFinalArray[1][eventName] gets the name of the second event, etc. This gets EVERYTHING that a user has done within the database. If you just want the user's point total, using searchUser() would be faster.

**`searchUser($dbPipeline, $searchUserId)`**

This function TAKES the database connection and a user id and RETURNS that user's first name, last name, and the total points they have right now. This function doesn't update the database with new information.

Example:
```
$myArray = searchUser($databaseConnection,
$idOfPersonYou'reLookingFor);
```

The array that is returned, $finalSearchUserArray, will have $finalSearchUserArray['firstName'] for the first name, $finalSearchUserArray['lastName'] for the last name, and $finalSearchUserArray['totalPoints'] for the total points. This function only returns one array, not an array of arrays. So accessing $myArray['firstName'] will return someone's first name. This function is the easiest way to get total points.

**`getPunchClock($dbPipeline, $getPunchClockId)`**

This function TAKES the database connection and a user id and RETURNS the list of punch ins and punch outs of that user in an array of arrays. This function doesn't update the database with new information.

Example:
```
$punchClockArray = getPunchClock($databaseConnection,
$idOfPersonYouWantThePunchClockInformationFor);
```

The array that is returned, $clockArray, will have $clockArray[timeIn] is "what time they checked in," $clockArray[timeOut] is "what time they checked out." Each of these arrays are contained within a larger array. So $clockArray[0][timeIn] is the first "timeIn", $clockArray[1][timeIn] is the second, and so on.

**`findUserId ($dbPipeline, $username)`**

This function TAKES the database connection and a username and RETURNS the id of that user. This function doesn't update the database with new information.

Example:
```
$userIdIWantToFind = findUserId($databaseConnection,
$usernameOfPersonIWantTheIdFor);
```

This function returns the user id as a string. You can call this function if you have a username but need the id for another function.

**`findEventId ($dbPipeline, $eventName)`**

This function TAKES the database connection and an event name and RETURNS the id of that event. This function doesn't update the database with new information.

Example:
```
$eventIdIWantToFind = findEventId($databaseConnection,
$eventNameIWantTheIdFor);
```

This function returns the user id as a string. You can call this function if you have an event name but need the id for another function.

**`login ($dbPipeline, $username, $password)`**

This function creates a HTML session for users. It is only useful with HTML.

**`logout($dbPipeline)`**

This also only works with the HTML session array. Only useful with HTML.

**`resetPassword ($dbPipeline, $username, $day, $month, $year, $newPassword)`**

This function TAKES the database connection, username of the user that forgot their password, day, month, and year of that person's date of birth, and the new password they want to input and RETURNS nothing. This function updates the database with a new password for one particular user.

**`updateEventPoints ($dbPipeline, $eventId, $newPoints)`**

This function TAKES the database connection, the id of an event, and the new point value that you want to assign to that event and RETURNS nothing. This function updates the database with a new value for an event.

Example:
```
updateEventPoints($databaseConnection, $eventId of event you
want to change, $new point value you want to assign to that
event);
```

This is a function that allows administrators to change the point value of an event. This will automatically update all of the points that everyone in the database earned from that event.

**`getMood ($dbPipeline, $userId)`**

This function TAKES the database connection and the id of a user and returns the history of that user's mood ratings. This function doesn't update the database.

Example:
```
$variable = getMood($databaseConnection,
$IdOfUserYouAreLookingFor);
```

The array that is returned, $moodArray, will have $moodArray[firstName] is the first name of the user you're looking for, $moodArray[lastName] is the last name of the user you're looking for, $moodArray[mood] is their mood for that instance, $moodArray[dateComplete] is the date of that mood, and $moodArray[journal] is the journal entry for that instance. Each of these arrays are contained within a larger array. So $moodArray[0][mood] is the first "mood", $moodArray[1][mood] is the second, and so on.

**Read Me File**
All the information below is available in readme.txt that will be included in the database package for the hand-off:

```
***********************************************/
* iHuman Moving the Mountain Database Project      *
*                                                  *
* Created by Grady Zielke & Kara Au                *
/**********************************************
```

As of April 16th, the final version of this iteration of the database can be viewed at "http://hucodev.srv.ualberta.ca/kwau/iHuman/index.php"

THE ABOVE LINKED ITERATION OF THE DATABASE IS SUBJECT TO REMOVAL WITHOUT NOTICE!!
It is a test database and should NOT be used as the official iHuman database.

```
*****************/
* INSTRUCTIONS *
/****************
```

**// EDITING DBCONNECT.PHP TO CONNECT TO YOUR MYSQL DATABASE //**
To do this, you require "dbconnect.php"

1. Open dbconnect.php in a simple text editor like Notepad. Do NOT use Microsoft Word.

2. All the values in CAPS must be edited with the correct database information. If this information is incorrect, you will have errors, and nothing will work:

**HOST NAME, USERNAME, PASSWORD, DATABASE NAME**

This is your MySQL database login information. Please consult whoever is in charge of your hosting for this information.

**// UPLOADING FILES FOR FIRST TIME USE //**
To do this, you require "index.php", "DBPrototype.php", "DBPrototypeLibrary.php", "dbconnect.php", "iHuman_database.sql", "stylesheet.css" and access to an FTP server.

1. ALL THESE FILES MUST BE IN THE SAME FOLDER. THEY SHOULD BE IN THE FOLDER "MTMdatabase".
Place the ENTIRE FOLDER on the FTP server. They should be on the FIRST LEVEL of the server and not housed within any other folders for proper redirection, or else you will have to edit link redirects within the php files manually.

NOTE: This is for proper redirection purposes. The current path for redirection is set to "MTMdatabase/index.php" etc. Please have a professional present to alter any link redirects.

**// SETTING UP THE DATABASE FOR FIRST TIME USE //**
To do this, you require "iHuman_database.sql" and access to MySQL.

1. If manually running, the query will be "SOURCE iHuman_database.sql;" WITH the semicolon in MySQL.

NOTE: The admin account should be automatically created at that point, with the login information of "admin" (username) and "password" (password) and "1991-11-11" (date of birth). The password may be reset in the reset password function that is visible on the login page after setup (with username and date of birth.

**// NAVIGATING TO THE WEBPAGE FOR FIRST TIME USE //**

1. Navigate to "yoursite.com/MTMdatabase/index.php" in your browser, where "yoursite.com" is the domain of your website. "index.php" is the login page of the database. You cannot access the other links until you have logged in. It will automatically redirect you to "index.php" if you navigate to "DBPrototype.php".

2. Ensure that all functions are properly running. You should be able to log in with the admin account credentials provided above. The admin is capable of adding and editing point values and events. Regular users should not be able to see this.

*********/
*** NOTES ***
/********

1. If the admin account creates an event, it can be accessed by all users.

2. Only the admin can allocate point values to events. All other user-created events will have a point value of 0 until changed.

***************************/
*** CODE DOCUMENTATION ***
/**************************

The documentation for the code is all located within "DBPrototypeLibrary.php". You can find the answers to how code works by opening that file in a text editor.

**Moving Forward**
We were in the process of implementing a secure password hash function for the database, but in the end, could not get it to work properly in testing and dropped it. It would be worthwhile to look at getting a professional IT firm to look at the security issues in the database and library in the future, especially if you implement the database as part of an online system or mobile app.

All installation information for the database can be seen in the readme.txt file that will be provided in the hand-off and above in this report.

Grady will be available for any issues that arise while combining the database and the front-end interface, and will be available for at the very least the rest of the summer.

**Additional Resources**
MySQL - [https://www.mysql.com/](https://www.mysql.com/)
This is the database software that we used to construct the database. It needs to be installed onto the server or machine that you want to host the database on in order to use it. mySQL is a fairly common database technology, so any server space that you buy may already have mySQL installed. Servers that you set up yourself will not have mySQL automatically installed, though.

PHP - [http://php.net/](http://php.net/)
This is the server-side code that we used for the database library. A PHP parser needs to be installed onto the server or machine that you want to host the database and libraries on in order to use it. PHP is a common server-side coding language, so any server space you buy may likely

have a PHP parser already installed. Servers that you set up yourself will not have a PHP parser automatically installed, though.