Cassell Robinson, Ada Killic, Sam Gassman

Phase 1 Report

System design:

The phase 1 system is based almost exactly on what is outlined in the design specifications, where functionality goals are almost essentially to complete or expand upon what is outlined in the specifications.

For example, each user is capable of modifying their account data to various defined extents as required -- we have expanded upon this in various directions to allow different user types to have greater control of what areas are modifiable. This works in the direction mentioned by the specifications for how System Administrators and Graduate Secretaries should be capable of viewing/modifying all data on the system. Regarding implementation details for how this is achieved, this is outlined below:

Our system is designed on the idea that only one user type should be able to perform a specific action (excluding simulating other users which is shared by Graduate Secretaries and System Administrators, which allows for this design type to exist). From that idea, we were able to condense multiple potential pages into a single file where we instead differentiate what should

be available to view or modify based on your user type. In this, for example, we can allow all grade modifications to exist from one page, because the only difference existing between a faculty member and another user type when editing grades is whether or not the user should be able to consistently modify grades after modification has already been made.

If this site were static, we would end up duplicating a number of functions, where there would only be slight modifications by user type. Thus, we have grouped identical processes to individual pages as shown above.

Below is the general description of the process of simulating a user:

Upon logging in to the website, a user is assigned session variables based off of their user permission level and user id. These credentials are then duplicated in a set of new session variables called view+x where x is either uid or type (for user type/permission level).
The original set of session variables are fixed, and may only be modified by logging in or out. This original set is required to determine who the user is, and what capabilities they should have. The duplicated set has a rather expanded additional functionality. Since advanced permission users are capable of viewing other user's data to edit it (whether it be register for classes, change personal information, modify grades, etc.), there must be some way to retain who the user is themselves, but also a way to keep track of who the logged in user is simulating.

As a result, as mentioned above, the “view” session variables, which are required for simulating other users are what are used to determine (for the most part), who may access a page. Since account simulation exists, it is rather irrelevant to most pages what your user type is, and more so important what the user type is of the account you are acting as. Just as we do not let faculty members register for classes, if a system administrator simulates a faculty member, they must now take on more limited capabilities until they return to their account -- there is no reason to let system capabilities merge simply because user simulation exists, if we did allow for merged capabilities between user types (say acting as a student and a faculty member), then the system would be nonfunctional, because the specifications do not allow for faculty to register for classes, or students to modify other grades.

This large process of simulating a user on the front end is as easy as viewing the “other accounts” page, acting as whatever user type one desires (although escalating privilege type is not allowed), performing whatever actions are desired (with potential expanded capabilities to what may be modified -- for example, students and faculty may only modify their permanent address in their personal info. Higher permission level user types are capable of modifying most user data), and then ending the simulation by selecting “back to my account”.

Schema design:

The schema of our database is based on the initial specifications from the REGS pdf, where our goals were to eliminate excessively duplicated data, as well as allow for fast access time to data.

Primarily, our schema provides efficiency by splitting duplicates from a number of tables into isolated tables such as users and student, as well as course, coursedata, lab, prereq, coreq.

The tables users and student are a good example of how we have moved to reduce extra data. Although it is certainly possible to store all user data in a single table, in doing so, since students contain extra data that is not found in any other user type, in a large database, merging the two tables would result in an excessive number of null values for non student users. Instead, rather than waste the space, by joining users with student on user id, we are able to reconstruct the data without any loss or spurious tuples.

We take this even further with our data on classes. Under the idea that classes could eventually contain multiple sections, where some data (say professor, class name, department, etc) would be duplicated for each section of the class, we split the class such that all data consistent between sections of a class is only listed once in a course table, and all data that varies between sections of a class is listed in course data. This decision has been made similarly with the decision to split users and student with the idea that with significant numbers of classes, the duplicate information stored is absolutely wasted. As well, to enhance the speed at which classes

are found, rather than have them only be identifiable by multiple columns, we have created a CRN of our own, which corresponds to the classes and is entirely unique.

In our schema, there are a number of functional dependencies simply due to the number of split tables. Our schema is in at least BCNF in all cases, as it only contains trivial functional dependencies as an extension from 3NF, and lacks transitive dependencies from 2NF. As well, all tables require that each non key attribute does require the whole key to identify it.

Application Logic:

Our website depends heavily on our session variable use. A significant amount of functionality is determined by a user's session variable "type" and "viewtype", and further discriminated by "uid" and "viewuid". Upon navigating to each page, our PHP validates that the user is of the correct viewtype to see the page. Then, based on their actual type, we determine what level the user should be able to interact with the page.

As well, our website contains a custom .htaccess file for our website specific subdirectory such that all non recognized urls are redirected to either the index or login page based on whether or not the user was already logged in.

Finally, to compensate for our large amount of user interaction on this site, we have employed a significant amount of form validation with regular expression matching, and in some cases removed open form entries altogether by replacing them with fixed dynamic form types like “select” with inner “option” tags.`