The Freshman Seminars Database Application: Design and Usage

Charles Duan
Harvard University, Computer Services Department

September 15, 2002

1 Introduction

The Freshman Seminars database application is designed to allow for the management of seminar information and student applications to those seminars. It provides an interface for students to browse through and search for freshman seminars and apply to those seminars online. It also allows for the members of the faculty to review student applications and evaluate their preferences. And finally, the application matches students as best as possible based on the students' rankings of each seminar and the faculty's preferences with regard to each student.

The application is designed to be easily maintainable, as much as possible through the use of online form pages, and carefully balanced between automation and person-driven control. The data is backed by the FAS Oracle servers, and information is drawn from the Registrar's office where available. The dynamic page content is written using the PHP scripting language.[1]

In this document, we will focus primarily on the design concepts of the database and the web application interface. We will also cover the steps necessary to maintain the integrity and validity of this data, both during the application period and between application cycles.

2 Database Overview

Essentially, the database is made up of four tables: the ***seminars*** table, the ***faculty*** table, the ***applicants*** table, and the ***applications*** table. The data in these tables also reference data provided from the Registrar's office, given in database views. The views provided are as follows:

ido.freshsem_course The Registrar's data on the seminars being offered for the year. This data is considered unauthoritative, and so the relevant information is essentially copied into the ***seminars*** table.

ido.freshsem_course_faculty_head The faculty members instructing the seminars, again retrieved from the Registrar's data. It is also considered unauthoritative, and it is copied into the ***faculty*** table.

[1]Available at *http://www.php.net*

ido.freshsem_freshmen_info Information on freshmen who may be applying to seminars. This data is considered up to date, and so it is used directly. It is referenced under the view named ***students***.

Please refer to Appendix A for a detailed description of the contents of each table.

Future modifications of the application system may require changes to the database. In general, adding fields to tables will not pose any problems. However, modifications to the currently existing table columns would require a reevaluation of the CGI script code. In particular, the scripts assume that numeric columns need not be translated into HTML-entity code. If numeric values such as the *course_id* field are made into character types, all uses of that field would require the proper translation into HTML or URL standard format.

2.1 The Students View

In the original design of the Freshman Seminars application, the ***students*** "table" was actually just an alias to the Registrar's data. However, a single student appeared who was *not* in the Registrar's database—and was thus unable to apply for seminars. Naturally, providing for this student proved to be a bit of a thought challenge.

The solution (which is admittedly a bit of a hack) was to create a new table named ***extra_students***, and change the ***students*** synonym to a view unioning the Registrar view and this new table. Of course, now, it is another table which needs to be cleared out on a yearly basis. Note, however, that this scheme is *not* implemented in the table generation script described in Section 5.1! If all of the tables and views are recreated in this database, and the same problem arises again, it will be necessary to reconstruct this table modification.

2.2 The Seminars Table

The ***seminars*** table contains information on each seminar. This data is imported from the Registrar at the beginning of the application cycle. However, since the Registrar's data is often out of date, it is not considered authoritative. As a result, the table contains a copy of all data obtained from the *ido.freshsem_course* view, but is modifiable so that it may deviate from it.

Seminars are keyed by their *course_id* field, which is identical to the field of the same name in the *ido.freshsem_course* view. This is done to ensure that the imported faculty data matches up with the courses. However, when adding new seminars, it is necessary to generate unique keys for them. Currently those keys are created by a cycling sequence of numbers between 0 and 1000, as it is expected that (1) it will never be necessary to add over a thousand new seminars and (2) the values in the *course_id* field of the Registrar data will remain higher than 1000.

The *semester* field is a number indicating the semester the course is offered; this is identical to the *trm_pat_num* field in the Registrar table. The *offered* column is either one or zero, indicating a logical Boolean value. By default, all seminars are offered. All other columns should be self-explanatory.

2.3 The Faculty Table

The ***faculty*** table contains a list of faculty members and the seminars which they are instructing. The table is drawn from data from the Registrar, but is independent of that data and fully modifiable. It is keyed by the ID number of the instructor and the seminar course ID number.

The contents of this table are used both for listing the names of seminar instructors on the student pages and for authenticating instructors on the faculty review page. The list of faculty members associated with a seminar is modifiable through the administrative pages only, so if a faculty member is left out, someone with administrative privileges must add the instructor to the appropriate seminar.

2.4 The Applicants Table

Students who are in the process of applying to seminars will have a record stored in the ***applicants*** table. This table holds per-student information which is not available through the ***students*** view, including the student's high school and contact information. The table is keyed according to Harvard ID number. Like the ***seminars*** table, it is necessary to ensure that a student listed in the ***applicants*** table is in the corresponding view as well.

This table also stores the assigned seminar for the student; that value is provided by the matching algorithm after the application evaluation process has completed. Once the students are notified of their semiar assignments, they will need to confirm or reject the assignment; this is stored in the *confirmed* field of the table. Currently, a NULL value indicates that no confirmation has been made yet; otherwise the decision is stored as a numerical boolean value.

At the end of every application cycle, all students should be flushed from this table. There is currently no restriction on students taking a second seminar in the following semester.

2.5 The Applications Table

Student applications to each seminar are stored in the ***applications*** table. Each student is permitted to have exactly one application per seminar, so the table is keyed by both student ID and seminar course ID number. It also stores the student's reported concentration, career plans, and essays, which are considered application-specific.

Two ranking values are also stored: the *student* rank for the seminar and the *seminar* ranking for the student. The ranking system is described in detail in the description of the matching algorithm. With regard to the database, however, a seminar ranking of zero indicates that the student is unranked (and thus ranked randomly but below all numerically ranked students); a student ranked with a NULL value has been denied from the seminar. A student may not have two applications to different seminars with identical rank; however, this is not enforced at the database level.

2.6 Communicating Updates: The News Interface

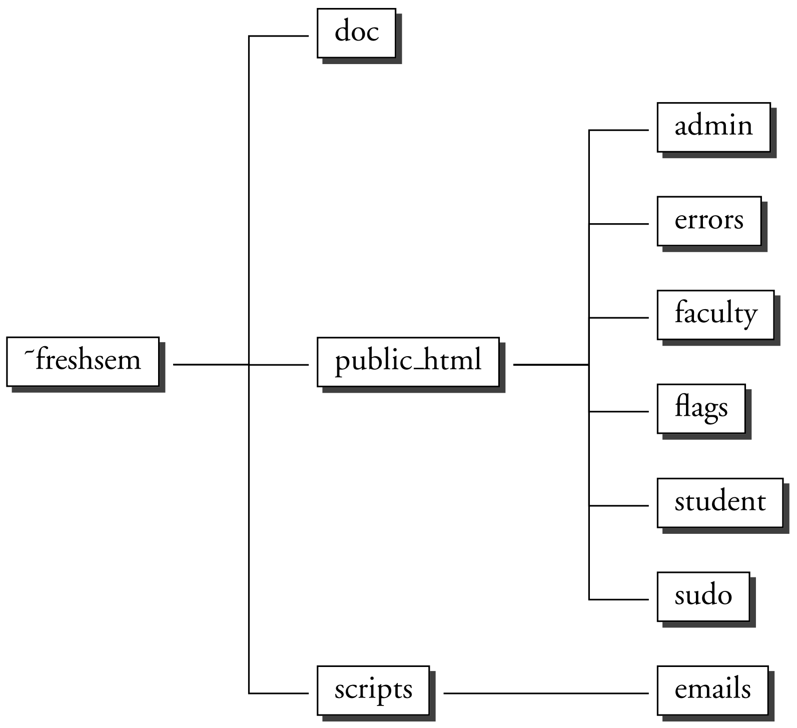
A simple news interface is provided for making small notifications about seminar updates. Each "article" consists of a timestamp and a short text string (500 characters or fewer). The news will be displayed in reverse chronological order (newest articles first).

Use of the news is at the discretion of the Seminar Office. Tools for creating and deleting articles are provided on the administrative web site.

3 File Organization

All of the files needed for the application are located under the ***freshsem*** account. Figure 1 shows the layout of the major organizational directories within the account directory. In general, these directories fall under two categories: directories containing viewable web pages, and directories containing scripts and other miscellaneous files.

Figure 1 Directory structure of the application system.



As described in Section 4, there are three major parts to the website: the administrative pages, the student pages, and the faculty pages. These correspond to the three main directories under the ***public_html/*** tree. The ***flags/*** directory contains due date flag files, described in Section 4.3. The directory named ***errors/*** contains the text of error-message pages, displayed when there is incorrect form data or other problems.

The contents of the ***sudo/*** directory could be put in the administrative directory, but I thought that it would be better to keep them separate. One possible advantage of this separation would be that people could be hired to enter data as proxies for students or faculty members, but they wouldn't have permission to change or view sensitive data through the administrative site.

The non-web-page directories are located immediately under the main home directory. The ***scripts/*** directory contains command-line scripts such as the matching algorithm; under it is a subdirectory named ***emails/*** which has the e-mailing script and its auxiliary files. The ***doc/*** directory contains this documentation.

4 Website Overview

The Freshman Seminars online application website is divided into three parts: the administrative interface, which provides for the management of seminars and applications; the student interface, where students apply for seminars; and the faculty interface, at which faculty members may review student applications. The individual pages are described in greater detail in Appendix B, which also contains webpage traversal graphs.

All of these sites contain a number of common elements. The two files ***sql.cgi*** and ***constants.php*** contain common functions, described below. All web pages displayed are written in PHP and include both of these files. Each directory contains the files ***header.php*** and ***footer.php***, which are included appropriately on every displayed web page. All of the pages also link to the stylesheet ***freshsem.css***.[2]

Additionally, a number of other conventions are kept in order to maintain consistency. Every student and faculty page stores the variables $id and $seminar, indicating respectively the University ID number of the person using the site and the *course_id* number of the currently selected seminar. This is important because they are used by those names in the header and footer files. Stylistically, pages generally follow the format outlined in Figure 2.

4.1 Development vs. Production

Any of the three web page directories can be set in "development" or "production" mode, depending on the existence of a file called ***.development*** in the directory. When in development mode, the web pages will use the Oracle development server *fasdv*; in production mode they will use the *fas* database.

As expected, the development servers should be used for testing pages and the production servers for real data input. The pages should not go live while still working in development status. To switch into development status, enter the directory and enter the command "touch .development" at the prompt. To go to production status, enter "rm .development".

The status flags only cause a distinction in the database used. Specifically, they do not cause a distinction in the due date flags, described below. As a result, if a due date flag is set or removed, it will be recognized by all pages, whether in development or production.

4.2 The Code Base: *sql.cgi* and *constants.php*

As mentioned earlier, the two files ***sql.cgi*** and ***constants.php*** form the major programming code basis for the Freshman Seminars application. There are five major query-related functions in sql.cgi, as shown in Table 1. The other file, constants.php, contains primarily functions used in determining the current semester and academic year. It also defines one function, *is_teaching()*, which determines if a professor with a given ID number is instructing a specified seminar. This, naturally, requires a database query, so it depends on sql.cgi. This, however, is the only dependency between them.

4.3 Due Date Flags

Rather than writing automatic due dates into the program that cut off at a specific time, due dates are implemented by the use of flags which may be manually turned on (or off). They may be toggled

[2]To ensure that they are all identical files, the copies of sql.cgi, constants.php, and freshsem.css are all hard-linked to each other.

Figure 2 Basic layout of a web page.

```
<?php

/*
 * filename
 *
 * Brief description of the file
 */

# Library inclusions/requires
require ’sql.cgi’;
...

# Filename aliases
$seminar_url = ’seminar.cgi’;
...

# Oracle updates and queries, algorithmic code
do_sql_query("UPDATE table SET ...");
...

?>
<html>
<head>
<link rel="stylesheet" type="text/css" href="freshsem.css">
<title>Page Title</title>
</head>
<body>
<?php include ’header.php’; ?>
<h1>Page Header Title</h1>
<p>Page body text...</p>

Results of a table query, form, etc. Limited PHP code may be
included, but only for retrieving data variables or SQL query
results.

<?php include ’footer.php’; ?>
</body>
</html>
<?php close_sql(); ?>
```

Table 1 Query functions provided through the file ***sql.cgi***.

do_sql_query(*string* [, *boolean*])
Performs an SQL query specified in *string*, and returns the statement handle. The *boolean* variable determines whether to use autocommit; if it is set to false the transaction may be committed using the function *commit_sql()* or canceled with *rollback_sql()*.

do_sql_pquery(*string*, *array* [, *boolean*])
Like *do_sql_query()*, but it allows for prepared statements. To do so, specify the query using bindings of the format ":bind" and use an associative array to match the bindings to variables, for the *array* parameter:

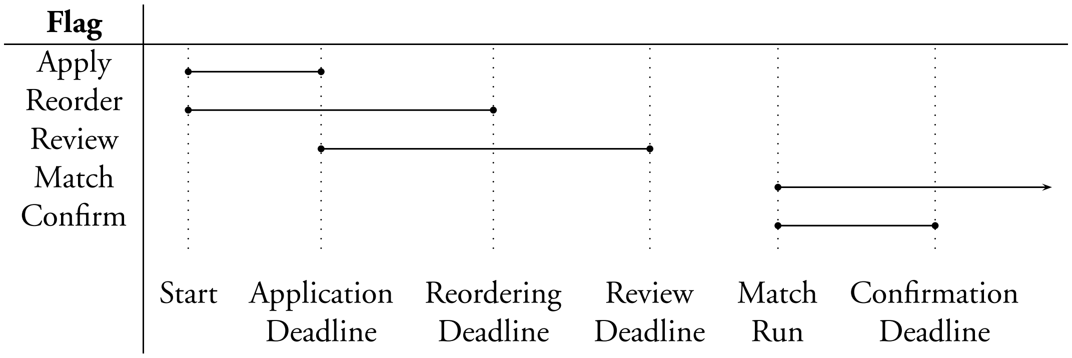
```
do_sql_pquery("SELECT * FROM x WHERE id = :id",
array(":id" => $id_var));
```

get_sql_line(*statement-handle*)
Takes as its parameter a statement handle returned by *do_sql_query()*, and returns the next row retrieved, in an ordered numerical array.

fetch_sql(*query*)
Performs a query and returns a single row retrieved from that query. The statement handle is not retrievable (a future implementation should destroy the handle).

pfetch_sql(*query*, *array*)
Is to *fetch_sql()* as *do_sql_pquery()* is to *do_sql_query()*.

Table 2 Timeline of deadline flag states.



through the administrative interface. Table 2 provides a timeline for the appropriate state of the flags corresponding to the various due dates within the application cycle.

These flags are in reality files in the ***flags/*** directory; the names of the flag files are defined in constants.php. In general, the existence of a flag file means that some action can be performed, so removing all the flags would prevent any changes from occurring (e.g. starting a new application cycle).

Within constants.php there are several functions defined (e.g., *can_apply()*) which test for the status of a flag (in this case, whether students may apply for seminars). They are advisory flags, so one must check for the appropriate flag explicitly before performing an action which may conflict with the appropriate status. In other words, if a web page modifies a student's application, it should check the *can_apply()* function first.

4.4 Administrative Authentication

The administrative sites are protected by PIN authentication, just like the rest of the site. The users permitted to view those pages are dependent on the directory which the pages are contained. Such protected directories contain a file named ***.admin***, which contains the ID numbers of those persons permitted to view the pages.

The authentication check is done in constants.php; it essentially tests for the existence of the file and then compares each line with the REMOTE_USER variable for a match. On success the page continues loading; on failure an error is displayed and the PHP interpreter exits. It is important to note that the check is performed *only* if the page includes constants.php.

4.5 Emulating Other People

Sometimes it is impossible for a student to fill out an application online or for a faculty member to enter their preferences for applications—or possibly the student or faculty member may just not want to use an online form. In that case, we provide the administrative users with a facility to "turn into" another person based on ID number and act on behalf of that person.

The utilities for doing so are contained in the ***sudo/*** directory, which provides a page for switching to and from users.[3] Implementation-wise, the user being emulated is stored as a cookie in the administrative user's browser.

When the cookie is set, the constants.php inclusion script runs a check to authenticate the actual user, based on the REMOTE_USER server variable and the ***sudoers*** file, which contains a newline-delimited list of the ID numbers which have permission to emulate other users. A failed authentication will exit with a short warning; upon success the REMOTE_USER variable will be changed appropriately to reflect the user being emulated. The cookie may be deleted manually through the main sudo page or automatically by closing the browser.

5 Application Cycle Walkthrough

As outlined by David Heitmeyer, the application cycle consists of six phases:

1. Data Input

[3]The name "sudo" is from a Unix command which has similar functionality.

2. Student Application Submission
3. Faculty Review
4. Algorithmic Matching
5. Student Acceptance
6. Freshman Seminar Office Matching

The application cycle will not be discussed; instead we will consider what operations must be done at periods during this cycle.

5.1 Initialization

Generally the table structure should be intact, so it should not be necessary to recreate the database every year. In case of special circumstances (e.g., physical data loss) the file ***tables.sql*** contains the SQL statements needed to recreate it.[4]

To import the Registrar's data into the database, execute the script ***import-seminars.php***. This should be done exactly once before the fall term application cycle. Because it imports data for both semesters, it need not be run before spring term. The script essentially copies values such as course numbers and titles out of the Registrar's database. If the data provided from the ***ido.freshsem_course*** or ***ido.freshsem_course_faculty_head*** should change, this script will need to be modified as well.

The Registrar does not provide course descriptions for seminars; these need to be imported separately. Because there has not been any standardized format for this data decided upon yet, there is no formal script for automatically importing this data. A program named ***import-desc.php*** can take a set of files, each named for the seminar course number and containing the description of the seminar, and insert them all into the database; however, this requires that the files be properly generated. Comments in the script indicate the proper mechanism for doing so.

5.2 Student Application Submission

Generally the database application will handle itself during the student submission phase. The only special cases arise when seminars are made not available, as there may already exist student applications to those seminars. In such cases, we may not wish to delete student applications immediately, as the students may wish to save a record of their application beforehand.

Each seminar has an "offered" flag which indicates whether or not the seminar is being offered. If a seminar is to be removed from the listing, this flag can simply be set through the administrative interface. Students will no longer be able to apply, but current applications will remain intact and viewable (but not editable, and warnings will appear to notify the student of the seminar's removal). Students with applications to seminars not offered should also be notified daily via e-mail; this is done by the ***notify-students.php*** program, which should be run regularly.

[4]The current implementation of tables.sql does not provide for adding in unregistered students. Please refer to Section 2.1 for directions on how to do so.

5.3 Faculty Review

Once the student submission period is over, the appropriate flag should be removed via the deadline controls, and the faculty review flag should be set on.[5] Students will still be able to reorder and/or delete their applications but not modify them; application-level controls will enforce that. During this overlap period faculty members should also be warned whenever student rankings are displayed that those rankings are tentative—in fact, since students may delete applications, the applications themselves should be considered tentative. Once the period of student application reordering is over and that flag is turned off, the faculty pages will display those preferences as being final.

We would assume, at this point, that the seminar data is finalized; that is, seminars will not be canceled at this point. The faculty review pages allow faculty members to review all seminars—even ones not being offered, although there should be no applications to them. The matching algorithm, of course, will ignore seminars not being offered and seminars offered in different semesters.

5.4 Algorithmic Matching and Student Notification

It would be a good idea, before running the algorithmic matching, to make a backup of the vital database information. At this point, the text of the applications does not need to be saved, but it is important to store the student ID, course ID, student ranking, and seminar ranking fields from the applications table, and the course number to course ID mapping from the seminars table. If the rest of the database is lost, these will provide sufficient information to run the matching process.

When the matching script is to be run, all flags should be off, indicating that no concurrent changes can be made to the database. The algorithm implementation is contained in a script named ***match-algorithm.php***; it is written in the PHP language and depends on sql.cgi for database interaction.[6] The script is run from the command line; no arguments are necessary. Most of the output of the program may be ignored.

Once it has been run, the administrative statistics page can be used to verify that a satisfactory matching job was done. Also, a backup should again be made, this time of the student ID and seminar fields in the applicants table. Once this is done, we may send out the results of the match by e-mail.

The mailing script is named ***email-results.pl***; it is written in Perl (since Perl has superior file-handling routines). This script reads in several template files listed as constants at the top of the script file; it also produces a log file of its actions. During testing, we don't actually want to e-mail students, so we have a variable $TEST_ACCOUNT set to the Freshman Seminars account; for the production run the actual addresses should be used, of course, which can be done by commenting out a line specified in the file.

The e-mail texts, of course, should contain links to the appropriate web pages, so that students may confirm their assignments.

[5]If both the student submission period *and* the faculty review flag are on, then the faculty pages are viewable, but changes may not be made to the data. This is mainly useful for testing and development purposes. Faculty members should *not* be provided access to student applications prior to the end of the submission period; this is not fair to students.

[6]Since it depends on sql.cgi, the .development flag can take effect by existing in the script directory. For the production run, ensure that the flag is off.

5.5 Student Acceptance

Students may choose to confirm or reject their seminar assignments once the matching algorithm has been run; this is denoted by making the appropriate changes on the deadline control panel. Students may then visit the web page listed in their e-mail.

Students who do not visit the web page and confirm their selection will be considered as not accepting their assignment (i.e., rejecting it). Since we keep track of students who have neither confirmed nor rejected their seminar assignment (stored as a NULL value in the database), we may change these semantics if desired.

After the deadline for confirming seminars has passed, the appropriate deadline flag in the administrative controls should be unchecked, thus allowing students who were not placed into any seminar to view the list of seminars with openings. At this point, no more changes to the database will be made, and the application cycle is considered finished. Further changes to seminar enrollment will go through the Freshman Seminars Office directly and not be reflected here.

5.6 Cleaning Up

When the seminar enrollments have stabilized and all the necessary statistics have been collected, we can begin to clean up the database. First, all of the flags should be turned off, as a security precaution. Archiving of the data can now be done as desired. Because the standard database tables must be empty at the start of each application cycle—except for the table of seminars, which must be cleaned yearly—the best method for intra-database archiving would be to create duplicate archive tables and move all of the records, adding a year field to each record.

6 Concluding Remarks

The Freshman Seminars application system is not too big a program, and there shouldn't be too much involved in maintaining it. Because it should be a relatively low-traffic web site, I've generally put security and consistency above performance, in order to ensure that everything works as smoothly as possible. Additionally, because the number of possible concurrent actions is very small, there isn't much of a risk of data problems, and production-time maintenance should (hopefully) be minimal.

That being said, this application was not designed to be entirely self-contained and maintenance-free. I have generally put preference in human-controlled actions where possible, such as in providing a page for deadline controls rather than making them automatic, and having the matching and notification scripts run from the command line. Because it's a new system right now, it should be used very carefully, and actions should all be monitored where possible. The only time-driven event I can think of, in fact, is the calculation of the academic year and current semester (in constants.php).

It's a good system; let's hope it serves its purpose for this year and many years to come.

Appendices

A Database Tables

Seminars

course_id *Primary key*
: The unique numeric identifier for each seminar. If a seminar was imported from the Registrar, then this value will be identical to their *course_id* value.

catalog_num
: The catalog number of the seminar, as listed in the *Courses of Instruction*. This (and other fields, unless otherwise noted) is imported from the Registrar where possible.

title
: The title of the seminar.

semester
: The semester in which the seminar is being offered. A value of one (1) corresponds to the fall term, two (2) corresponds to spring, and three (3) to summer.

list
: The long description of the course. This text field should also contain any additional questions which the course instructor would like students to answer (see column *applications.essay3*). The seminar description is stored as HTML-formatted text, so it is possible for a malicious seminar description to break the application. However, since we expect that this data is coming from a trusted source, we will accept that.

capacity
: The number of students permitted in this seminar. Default is twelve (12).

offered
: Boolean value for whether the course will be offered. A value of zero (0) indicates that it will not be offered; one (1) means that it will be offered. Default is 1.

course_num
: The course number (e.g. 32a) that is used in the *Courses of Instruction*. This number is imported from the Registrar by concatenating the *num_int* and *num_char* fields in the ***ido.freshsem_course*** view.

Faculty

course_id *Primary key (with huid), references seminars.course_id*
: The reference to the seminar which this faculty member is instructing.

huid *Primary key (with course_id)*
: The ID number of the course instructor.

name
The full name of the course instructor.

Applicants

huid *Primary key*
The ID number of the student. This currently does not reference the ***students*** view from the Registrar (since doing so would affect the Registrar's database).

contact_info
Any additional contact information the student wished to provide.

highschool
The student's reported high school.

highschool_loc
The location of the student's high school.

seminar *References seminars.course_id*
The seminar to which the student has been assigned. This value should remain NULL until assigned by the Algorithmic Matching phase or modified manually afterward.

confirmed
Whether or not the student has confirmed his/her seminar assignment. This value is NULL if no decision has been made, one (1) if affirmative, and zero (0) otherwise.

Applications

huid *Primary key (with course_id), references applicants.huid*
The ID number of the student owning this application.

course_id *Primary key (with huid), references seminars.course_id*
The reference to the seminar toward which this application is directed.

st_rank
The student's ranking of this seminar. It should be a value between 1 and 3 (or the maximum number of applications which a student is permitted), where 1 is highest.

sem_rank
The course instructor's ranking of this application. This may be an arbitrary integer value up to 999. A positive value (greater than 1) indicates a ranking, with 1 being highest; a value of zero indicates no preference, ranked below all applications with numeric rank. A NULL value means that the student has been denied from the seminar.

random
A random value assigned to the application, used to provide a ranking for unranked seminars (i.e., when *sem_rank* is zero).

concentration
The student's concentration on this application.

career_plans
The student's career plans.

essay1
The first essay response.

essay2
The second essay response.

essay3
The third essay response. This is the optional response; the prompt for it should appear in the course description (the *seminars.description* column).

News

newsdate
The date of the posting.

info
The text of the posting.

B Web Pages

Pages not marked with a *reached from* or *leads to* clause have multiple and options in those directions; they are generally not crucial links. Figures 3, 4 and 5 show the major traversal paths through each website division; in those graphs, dashed lines represent pages which are loaded within another script, dependent on status conditions described in the text below.

Administrative Pages

add-faculty.cgi *Reached from: edit-seminar.cgi; leads to: edit-seminar.cgi*
Form for adding a new faculty member to a seminar.

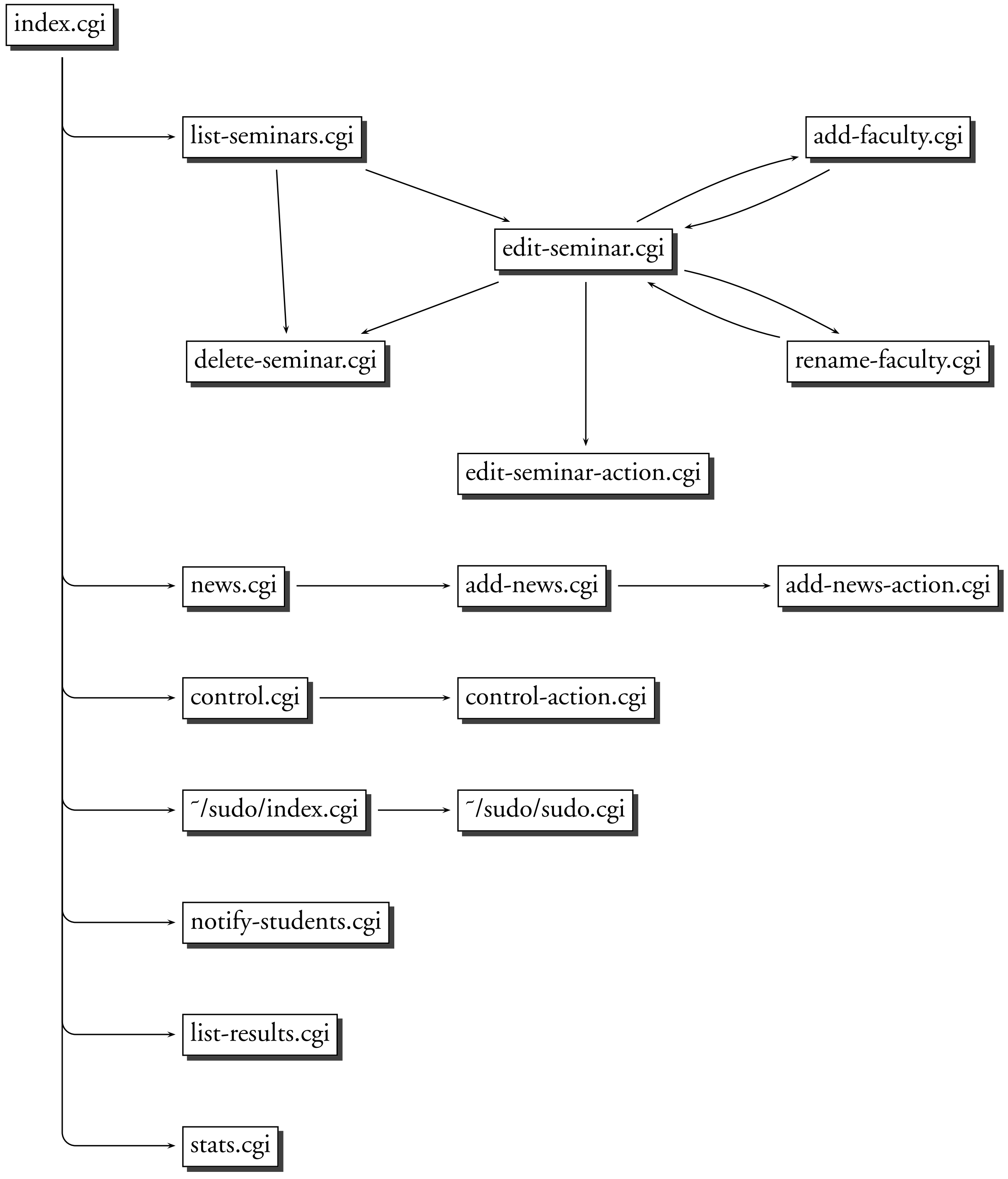
add-news-action.cgi *Reached from: add-news.cgi*
Inserts a news item into the database.

add-news.cgi *Leads to: add-news-action.cgi*
Form for adding a news item.

control-action.cgi *Reached from: control.cgi*
Updates the deadline control flags.

control.cgi *Leads to: control-action.cgi*
Form for updating the deadline control flags.

Figure 3 Administrative pages.



delete-seminar.cgi *Reached from: list-seminars.cgi, sem-detail.cgi*

Deletes a seminar from the database.

edit-seminar-action.cgi *Reached from: edit-seminar.cgi*

Updates the database with updated seminar information.

edit-seminar.cgi *Leads to: edit-seminar-action.cgi, delete-faculty.cgi, rename-faculty.cgi*

Form for modifying information on a seminar. It is also used to create new seminars.

index.cgi

Index page.

list-results.cgi

Lists all students and their assigned seminars, after the matching algorithm has been run.

list-seminars.cgi *Leads to: edit-seminar.cgi, sem-detail.cgi*

Short list of all offered seminars.

news.cgi

Lists all news items.

notify-students.cgi

Provides a list of all students who have applied to seminars which are not being offered. The students on this list should be notified by e-mail.

rename-faculty.cgi *Reached from: edit-seminar.cgi; leads to: edit-seminars.cgi*

Form for renaming a faculty member.

sem-detail.cgi *Leads to: edit-seminar.cgi*

Shows a full description of a seminar, in the format which students will see as well.

stats.cgi

Provides statistics on applications.

~/sudo/index.cgi *Leads to: ~/sudo/sudo.cgi*

Form for setting the ID number of a student or faculty member to emulate.

~/sudo/sudo.cgi *Reached from: ~/sudo/index.cgi*

Sets the ID number of the person being emulated, by creating or updating the cookie.

Student Pages

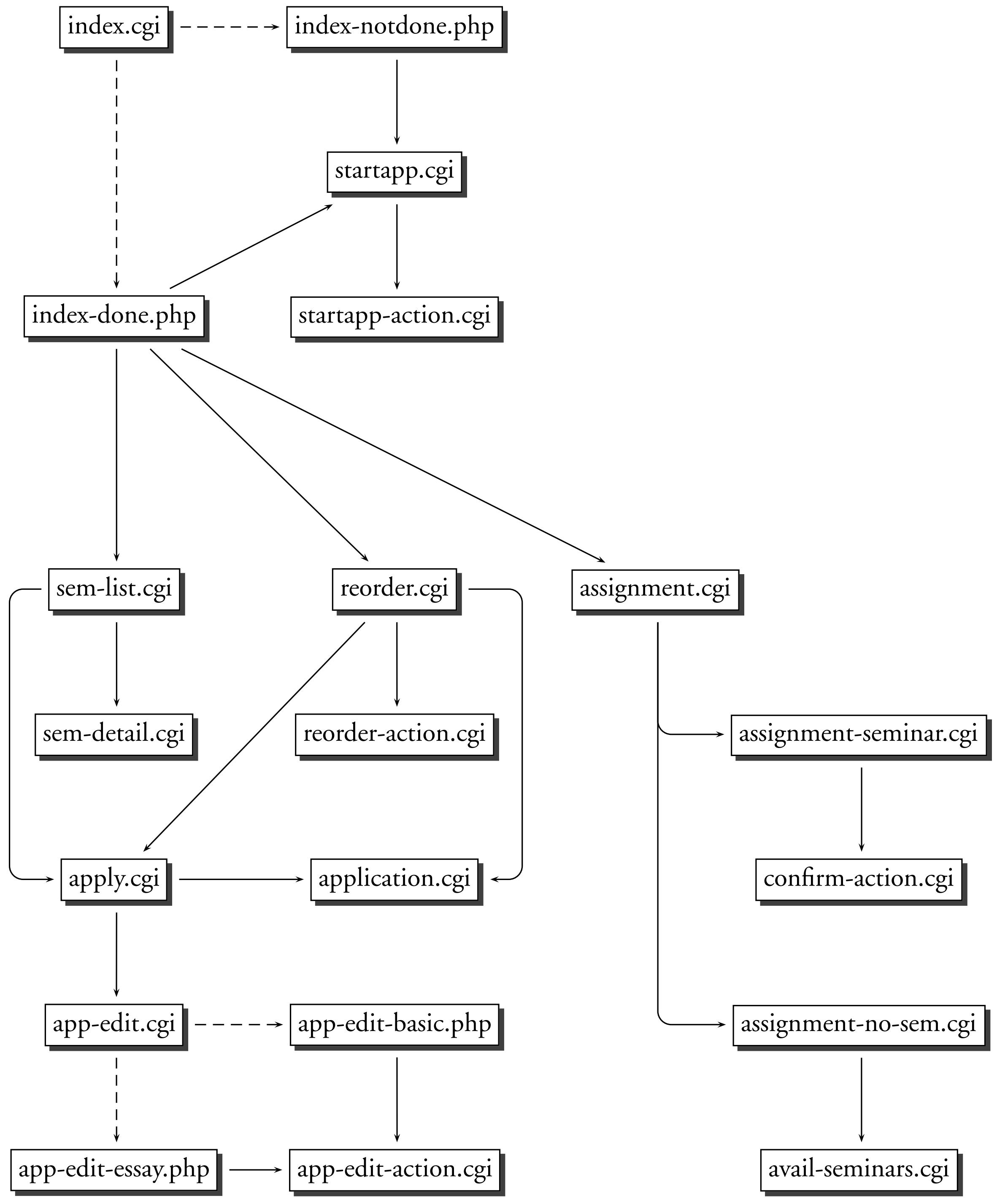
app-edit-action.cgi *Reached from: app-edit.cgi*

Updates the database reflecting a change in a part of an application.

app-edit-basic.php *Displayed on: app-edit.cgi*

Form for editing the non-essay fields of a seminar application (i.e., the concentration and career plan fields).

Figure 4 Student pages.



app-edit-essay.php *Displayed on: app-edit.cgi, displays: essay-*n*.php*
Form for editing an essay for an application.

app-edit.cgi *Displays: app-edit-basic.php or app-edit-essay.php; reached from: apply.cgi; leads to: app-edit-action.cgi*
Edits a part of an application.

application.cgi *Reached from: reorder.cgi, apply.cgi*
Displays, in printable format, the entire application to a seminar.

apply.cgi *Leads to: application.cgi, app-edit.cgi*
Interface for editing (or creating) an application to a seminar; provides links to pages which edit a part of the application.

assignment-no-sem.php *Displayed on: assignment.cgi, leads to: avail-seminars.cgi*
Seminar assignment results page, displayed when the student was not assigned to any seminar.

assignment-sem.php *Displayed on: assignment.cgi, leads to: confirm-action.cgi*
Seminar assignment results page, displayed when the student was accepted into a seminar; prompts the student to confirm or reject that assignment.

assignment.cgi *Displays: assignment-no-sem.php, assignment-sem.php*
Displays the student's assigned seminar, after the matching algorithm has been run.

avail-seminars.cgi *Reached from: assignment.cgi*
Lists seminars which have available spaces, after the student confirmation period is over.

confirm-action.cgi *Reached from: assignment.cgi*
Updates the database reflecting the student's choice to confirm or reject their seminar assignment.

essay-*n*.php *Displayed on: app-edit-essay.php*
The text of the *n*th essay prompt, where *n* is between 1 and 3.

index-done.php *Displayed on: index.cgi*
Index page, displayed when a student has started the application process; provides links to common application tasks.

index-notdone.php *Displayed on: index.cgi; leads to: startapp.cgi*
Index page, displayed when a student has not yet started the application process; prompts for the student to enter personal information on the appropriate page.

index.cgi *Displays: index-done.php or index-notdone.php*
Index page; the page displayed depends on whether the student has already started the application process.

news.php *Displayed on: index-done.php*
Inclusion file for displaying news entries.

reorder-action.cgi *Reached from: reorder.cgi*
Updates the database upon a reordering or deleting of applications.

reorder.cgi *Leads to: reorder-action.cgi, apply.cgi, application.cgi*
Provides for the management of currently open applications, with links to edit and view applications, as well as an interface for reordering and deleting applications.

search.cgi *Reached from: index.cgi*
Displays results of a seminar search.

sem-detail.cgi
Provides detailed information on a single seminar.

sem-list.cgi
Lists the seminars available.

startapp-action.cgi *Reached from: startapp.cgi*
Updates the database on changes to personal information (in the ***applicants*** table).

startapp.cgi *Leads to: startapp-action.cgi*
Provides for changing personal, non-application-specific information, stored in the ***applicants*** table.

Faculty Pages

apps-print.cgi
Provides the entire application forms of all students who have applied to the seminar, in printable format (using CSS2 formatting).

deny-action.cgi *Reached from: deny-list.cgi*
Updates the database reflecting decisions on denying students admission to the seminar.

deny-list.cgi *Leads to: deny-action.cgi*
Form for selecting which students are to be accepted or denied from the seminar.

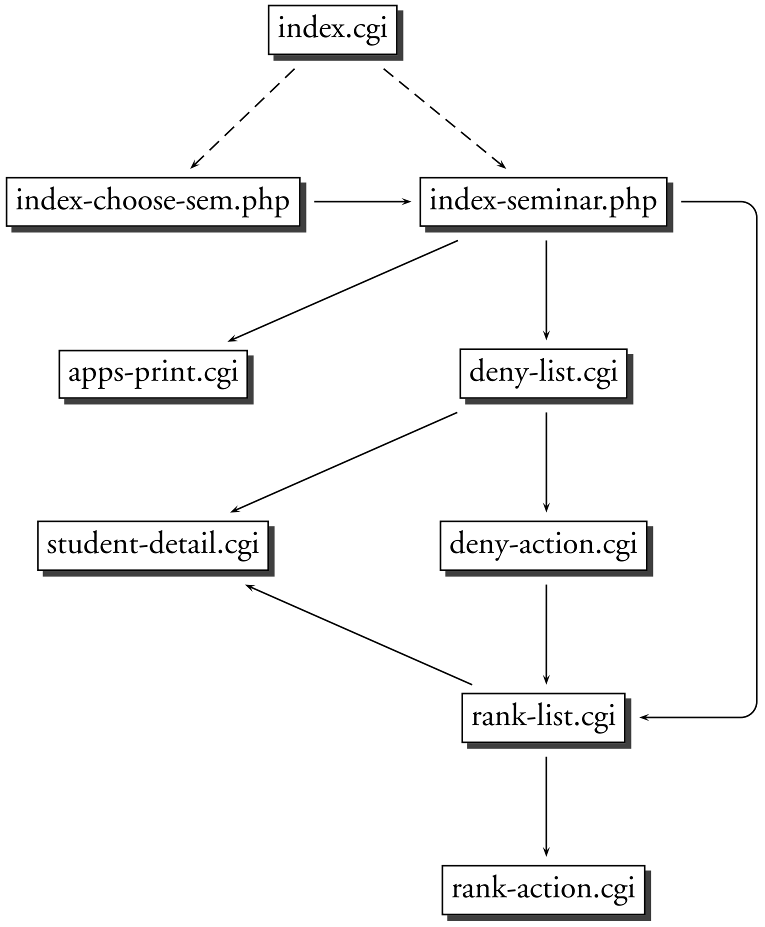
index-choose-sem.php *Displayed on: index.cgi*
Index page, when a seminar has not been chosen yet, prompting the instructor to select a seminar to review.

index-seminar.php *Displayed on: index.cgi*
Index page, when a seminar has been chosen, providing actions for reviewing that seminar.

index.cgi *Displays: index-choose-sem.php or index-seminar.php*
Index page; the page displayed is chosen based on whether a seminar has been selected by the instructor yet.

rank-action.cgi *Reached from: rank-action.cgi*
Updates the database reflecting decisions on student rankings.

Figure 5 Faculty pages.



rank-list.cgi *Leads to: rank-action.cgi*
Form for entering rankings for students.

student-detail.cgi
Provides a single student's entire application.

student-list.cgi
Provides a list of students who have been assigned to and who have confirmed their assignment to the seminar.