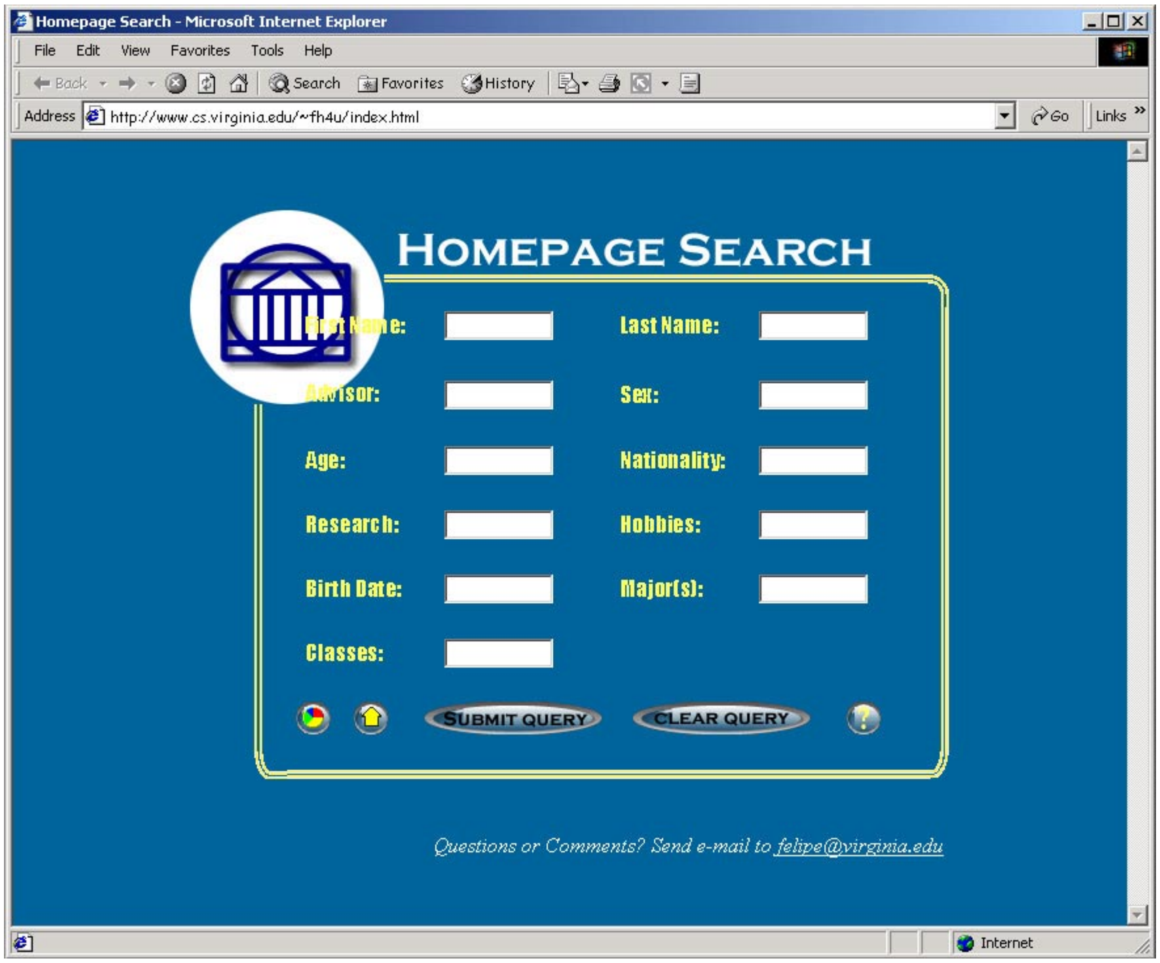
Frontispiece



http://www.cs.virginia.edu/~fh4u/index.html

Questions or Comments? Send e-mail to *felipe@virginia.edu*

A Database-backed Personal Information System for Automatic Creation of Home and Summary Web Pages

A Thesis
In TCC 402

Presented to

The Faculty of the
School of Engineering and Applied Science
University of Virginia

In Partial Fulfillment

of the Requirements for the Degree

Bachelor of Science in Computer Science

by

Felipe Huici
March 30th, 2001

On my honor as a student, on this assignment I have neither given nor received unauthorized aid as defined by the Honor Guidelines for Papers in TCC Courses.

Approved ____ (Technical Advisor)
David Evans (Signature)

Approved ____ (TCC Advisor)
Peter Norton (Signature)

Table of Contents

Glossary of Terms

.plan file: Text file containing a student’s personal information.

Cronjob: A particular task within a crontab file.

Crontab File: A task scheduler in the form of a text file that allows programs to be run automatically at regular intervals.

HTML: Hypertext Markup Language, the authoring language used to create documents on the World Wide Web.

JavaScript: A scripting language developed by Netscape to enable Web authors to design interactive sites

MySQL: A relational database management system.

Perl: A general-purpose programming language.

PHP: A language the goal of which is to allow Web developers to write dynamically generated pages quickly.

Script: An executable file containing a series of commands; a program.

SQL: Structured Query Language, allows users to access data in relational database management systems.

UNIX shell: A command language interpreter, the primary purpose of which is to translate command lines typed at a terminal into system actions.

URL: Universal Resource Locator, the global address of documents and other resources on the World Wide Web.

Abstract

Like most academic communities, Computer Science graduate students at the University of Virginia need to share personal information. Unfortunately, until recently there was no simple or quick way of doing this. A student could provide a text file in his or her home directory, but only users with accounts to the Computer Science server could view it; alternatively, a student could create a home page, but this was a very time-consuming. To solve this, I have implemented a system that reduces effort and time and offers a simple means of accessing the information. Using shell scripts, Perl, PHP, and MySQL, the system collects the information from students' text files and deposits it in a database. Scripts then allow anyone with access to the Internet to view automatically generated home pages, and to search and summarize the information in them. I concluded that a system based on database-backed home pages saves time, allowing a user to have a home page up in literally minutes, and provides a wide-reaching medium for information sharing.

Chapter 1: Sharing Personal Information Through Home Pages

Home pages are a wonderful communications tool; they can concisely display a portrait of their creators, and reach anyone with access to the Internet.   Yet home page creation is cumbersome, and Graduate students in the Computer Science Department at the University of Virginia currently have to cope with the tedious task of creating their home pages. Indeed, a visit to the department's web page reveals that some students do not even bother to create a home page. To make matters worse, finding information about an individual without one is cumbersome. A student must own an account on the Computer Science server, somehow find the target student's home directory, and view a file (usually called .plan) containing that information. Further, there is no simple way of searching or summarizing the information in these files: If I wanted to find the names of all students who have a certain professor as their advisor, or wanted to count the number of female students in the department, I would have to manually sift through *all* home directories. My thesis provides a solution to these problems through database-backed home pages.

1.1 Review of Relevant Literature

The creation of useful and efficient database-backed home pages has required the merging of several technologies and several years. Fortunately, these years have not been spent in vain, and today a simple site of this type can be set up with moderate effort, providing the powerful combination of abundant information and easy access.

The Internet makes this type of site feasible. The architecture for the Internet developed from the ARPANET, an experimental packet-switched network funded by the Advanced Research Projects Agency (ARPA). This preliminary network along with its two main protocols, TCP (Transmission Control Protocol) and IP (Internet Protocol), have become the Internet. Its extensive reach not only makes database-backed home pages possible, but also turns them into powerful avenues for gathering information.

Certainly this type of system could not function without an operating system, and one of the oldest is UNIX. Ken Thompson and Patrick Wood originally developed the UNIX operating system in the late 1960s. Their primary goals included the construction of an environment that permitted easy program development, and the creation of a small, easily maintainable, and memory-efficient operating system. But perhaps their greatest achievement was the development of a version of UNIX that could be ported (transferred) to different computer systems. This flexibility ended the previous routine of having to learn a unique operating system for each computer system, and, consequently, UNIX grew popular. The Computer Science department at the University of Virginia has many systems running UNIX, making it the operating system of choice for development.

The system lacks one last ingredient: the database. This field has seen great changes with the advent of relatively inexpensive database software. Historically the cost of databases was often prohibitive, mostly because of the hardware needed to run them with acceptable performance. Large and expensive software such as Oracle still exist, but as a result of the Open Source movement a few cheap options have emerged, including MySQL. Michael Widenius began the creation of MySQL with the development of the UNIREG database tool for the Swedish company TcX in 1979.

Dissatisfied with the existing technology, TcX and Widenius started working on a new project, and, as a result, MySQL 3.11.1 was released in 1996 for the Linux and Solaris platforms.

Connecting the web browser to the database and vice versa requires programming and scripting languages. In 1987 Larry Wall created a programming language, Perl, that adeptly brings databases and web sites together. Pierce explains that "Perl is used in so many places because Perl is what's known as a *glue language*. A glue language is used to bind things together." A scripting language with similar capabilities but a shorter lifetime is PHP (Personal Home Page Tools). Rasmus Lerdorf, its creator, initially created a number of tools along with a parsing engine. His efforts resulted in the release of PHP / FI. By 1997, more than 50,000 web sites were using this language for a wide range of applications, including database interaction and the display of dynamic content. Thus, PHP and Perl, along with a web server and HTML**,** are the last elements needed to implement a system for database-backed home pages.

1.2 Justification and Objectives

Computer Science students at the University of Virginia need to share personal information with each other, and there is no simple way to achieve this. Thus, there is a pressing need for quick creation of home pages, a way to search them, and a way to summarize the information in them. Students should be able to create a home page in the time it takes to fill out a text file with their personal information. They should have a home page in minutes.

Students should also be able to search the information on these pages quickly. For instance, a student should be able to look for peers that who in his or her office area, or peers from India who are 24 years of age. This interface should be accessible and easy to use.

Further, students should be able to view summaries of the information contained in the home pages: How many people are 24 years of age? How many are from India? How many have a particular professor as their advisor?

1.3 Solution

A student begins creating his page by filling a text file called .plan with his personal information. The format rules are, in general, simple: for each line, write the name of the field, a colon, and the value for that field. For instance, if a student is 22 years old, he would add the following line to his or her .plan file:

```
Age: 22
```

Notice that this line contains a space between the colon and the number; in fact, users can type each line in many different ways. I have designed the system to tolerate these discrepancies. The system requires, however, that the field name (the text to the left of the colon) match a predetermined list that will be distributed to all users.

Once the student types this file and saves it, his or her part is done. The system then takes care of recognizing that the file has been changed, and calls a script to enter this information into the database. If the file already existed but the student made changes to it, the system will update the information. Next, if the student wishes to view his home page, he can do so by accessing a web site using Internet Explorer or Netscape, entering his UVA id, and hitting enter. A PHP script will then display his newly created home page. Thus, a user could have a home page up and running in one or two minutes.

Searching is also simple. A student would go to the same address and type his query into the appropriate fields. For example, if I wanted to form a research group in the field of programming languages, I could type "programming languages" in the research text field of the web page. A PHP script will output all students with similar interests, with links to their home pages. Hitting enter without filling any text fields will cause the script to display all entries in the database.

Finally, to view a summary page, users go to the same address, select the summary, and another PHP script will display the results.

1.4 Possible Complications

The biggest barrier consisted of implementing a tolerant system. Since text files are loosely formatted, I had to create a system capable of distinguishing between data and extraneous characters like leading spaces and tabs. Further, students might write the same field name in different ways; e-mail might be written as both "e-mail" and "email." The system had to be able to recognize these two forms. Even field values such as dates can be entered in different formats, so I had to ensure that all forms were recognized.

It remains to be seen if the benefits of the system are enough to convince students to take the time to fill out their personal information files. Thankfully, many students already have done this, which should encourage others to follow suit.

Some students may feel uncomfortable about having their personal information publicly available on the Internet and, as a result, may be hesitant to use this system. I will address this concern by sending a privacy statement to all users assuring them that their information will only be shown in personal home pages, searches, and summary pages. Those students who feel that this measure is not enough can refrain from using the system by removing any personal information from their .plan files. Alternatively, if a student wishes to keep the information in the file but does not wish that information to be available on the Internet, I can manually erase their home directory from the system so that the system will never process the file again.

1.5 Overview of Technical Report

In the body I begin by providing a comprehensive description of the design, the rationale for design decisions, and a diagram of the overall design to clarify the discussion. I cover the design of the database, the design of the shell and Perl scripts in charge of input, and the design of the PHP scripts in charge of output to web pages.

The last chapter contains my conclusions, including system correctness, an analysis of the extent to which the solution solves the stated problem, and a section that discusses possible extensions to this project.

Chapter 2: System Overview

This chapter lays the basic groundwork so that the reader can easily understand the chapters that follow it. It is a bird's eye view of the design, discussing the database and scripting languages used, the pieces that make up the system, and how these pieces fit together.

2.1 Database and Scripting Languages

I chose MySQL for several reasons. First, it is quick. In fact, its developers claim that it is about the fastest database on the market. The MySQL benchmark page (http://www.mysql.com/benchmark.html) gives strong evidence supporting this claim: MySQL beats other databases like PostgreSQL, Informix, Access 2000, and Oracle in almost every category tested.

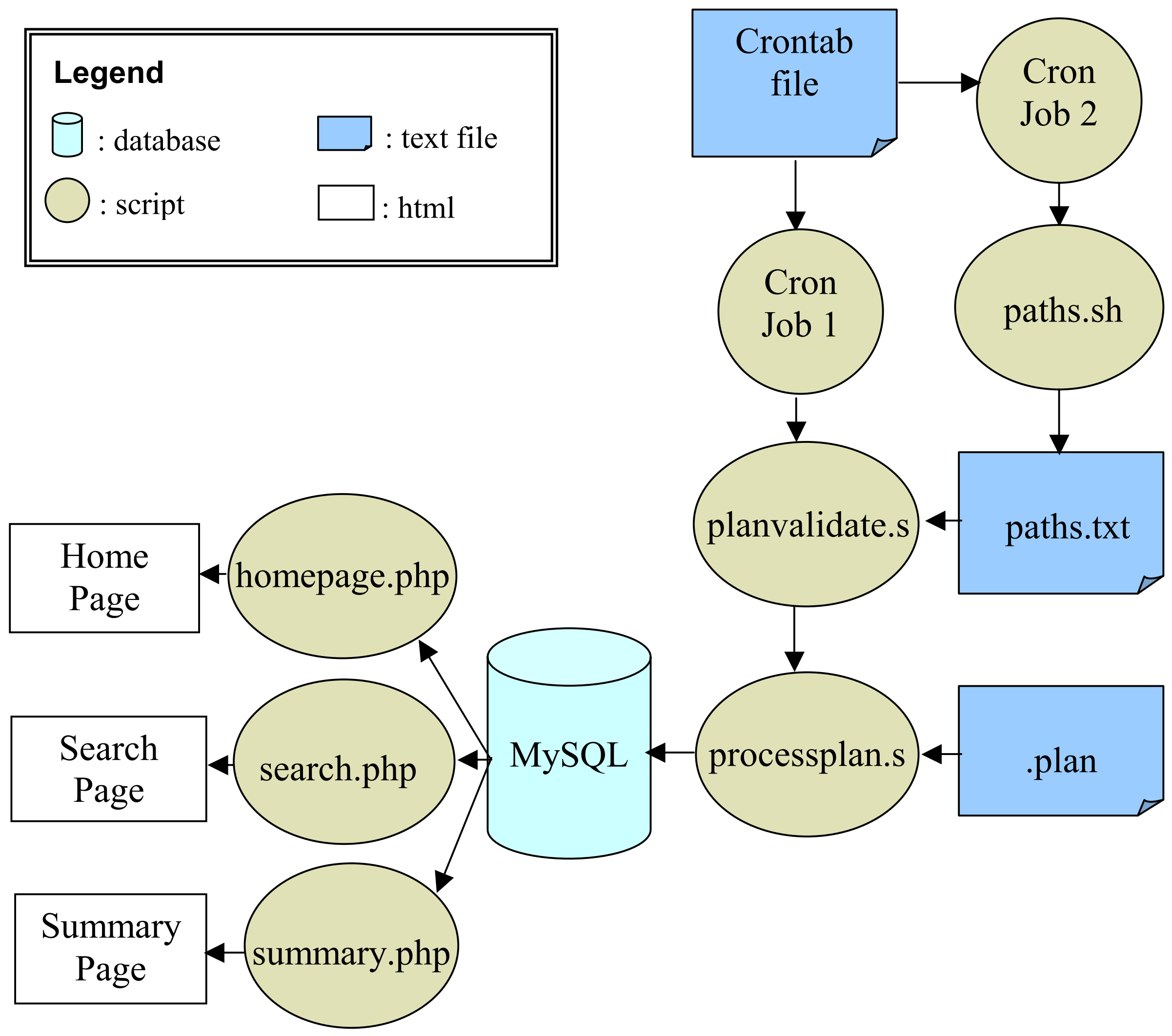
In addition, the MySQL server can be used freely for non-Windows platforms, a clear advantage over products like Oracle. MySQL also has a variety of programming interfaces for languages such as C, Perl, Java, and PHP, and its distribution is open: the program and its source code can be downloaded using a web browser. Finally, extensive technical support exists for MySQL, including a comprehensive reference manual, technical support contracts from the developers, and an active mailing list.

I selected Perl because of its pattern-matching capabilities. Recognizing patterns in text is fundamental to processing the students' personal information files. Further, Perl interacts easily with MySQL and is open source. I chose PHP because it can access

MySQL to generate dynamic content and create a web interface for searching elements in the database.

2.2 Explanation of Overall Design

The following figure gives a graphical description of all the elements of the system and how they fit together:



I will begin explaining the diagram from the right side, the input, and work towards the left side, the output. The process begins with the crontab file. A crontab file is a task scheduler in the form of a text file; it allows programs to be run automatically at regular intervals. In this case, the file consists of two lines, or cronjobs, each calling separate shell scripts daily (note that shell scripts have a "sh" extension, Perl scripts a "pl" extension, and PHP scripts a "php3" extension). The script paths.sh uses the commands `ypcat group` and `ypcat passwd` to obtain the paths to students' directories. To clarify, here is sample output from the first command:

```
csgrad:*:26:virt,cld9h,jdh8d
ugrad:*:35:
```

This sample says that Computer Science Graduate students have a code number of 26. The script uses this number to distinguish between paths of these students from paths of other users with accounts on the Computer Science server. Output from the second command follows:

```
71658:26:Yannick Loitiere:/uf6/ycl2r:/usr/cs/bin/bash
68798:26:Vinod Balakrishnan:/af4/vkb3q:/usr/cs/bin/bash
54038:111:Fritz Knabe:/af1/knabe:/usr/cs/bin/bash
```

Since the first two lines contain the code 26, the script writes the paths /uf6/ycl2r and /af4/vkb3q to the file paths.txt.

Next, planvalidate.sh verifies that the path and .plan files exist and are readable for each path in paths.txt. If that is the case, the script checks that the file has been modified in the last 24 hours by using the `stat` command:

```
: /uf2/fh4u ; stat .plan
File: ".plan"

Access: Wed Mar 21 11:47:44 2001 (00000d 00h 00m 26s)
Modify: Wed Mar 21 11:48:05 2001 (00000d 00h 00m 05s)
Change: Wed Mar 21 11:48:05 2001 (00000d 00h 00m 05s)
```

In this example, the .plan file has been changed recently, so planvalidate.sh would call the processplan.pl script with the path /uf2/fh4u as the parameter.

The Perl script processplan.pl is then responsible for processing the .plan file and inserting the data into MySQL. I will defer further discussion of this step until chapter 4. Once the information is in the database, it is PHP's job to output it. The system contains three scripts: homepage.php3 displays a student's homepage, search.php3 displays the results of a search, and summary.php3 displays a summary page. Users reach these scripts through the graphical interface provided by the web page index.html. I defer further discussion of system output until chapter 5.

Chapter 3: Database Design

This chapter discusses the tables in the MySQL database, how they are related, and what the unique keys are. I created the database using the Perl script createdb.pl. The last section of the chapter talks about user accounts, and why two types of accounts are needed.

3.1 Tables, Unique Keys

The system recognizes the following fields as valid:

```
office              date_of_birth
advisor             picture
fax                 extra_html
address             major
sex                 current_classes
age                 last_name
nationality         first_name
quote               e-mail
research            phone
hobbies
```

From these fields I decided to create three tables: main, phones, and emails, shown on the next page. The phones and emails tables exist because their respective fields exhibit a one-to-many relationship: one student can have many emails or phone numbers. Though there are other fields, such as current classes, that could have warranted their own tables, I decided to keep them in table main. This was done to make the database more efficient and because it is not as important to keep these fields separate when the system displays summary pages and performs searches.

The unique key for table main is id, which is auto-increment and cannot, therefore, be null. I use precisely this field as a reference into the phone and email tables. For instance, if I wanted to find the phone number of a student, I would search his or her

Table phone

Field	Type	Null
id	int(10)	No
first_name	varchar(20)	No
last_name	varchar(20)	No
office	varchar(30)	Yes
advisor	varchar(20)	Yes
fax	varchar(20)	Yes
address	varchar(200)	Yes
sex	enum('M','F')	Yes
age	int(10)	Yes
nationality	varchar(20)	Yes
research	varchar(200)	Yes
hobbies	varchar(200)	Yes
date_of_birth	date	Yes
picture	varchar(50)	Yes
extra_HTML	varchar(50)	Yes
major	varchar(40)	Yes
current_classes	varchar(75)	Yes
quote	varchar(255)	Yes
uvaid	varchar(15)	No

Table phone

Field	Type	Null
id	int(10)	No
phone_txt	varchar(20)	No
description_txt	varchar(15)	Yes

Table email

Field	Type	Null
id	int(10)	No
email_txt	varchar(30)	No
description_txt	varchar(15)	Yes

entry in table main, retrieve the id number, and read the phone number of the entry or entries whose id matched the id of that student. For the email and phone tables, the unique key cannot be the id, since several entries with the same id can exist in them. Consequently, the unique keys are the pairs (id, phone_txt) and (id, email_txt), meaning that a phone number and an email of a particular student are unique.

Note that the tables contain several fields that cannot be null. First, table main does not allow a student's name to be empty; it would not make much sense to accept a personal information file that lacks the owner's name. Clearly primary keys have to be filled, resulting in the fields id, phone_txt, and email_txt being not null. Further, the uvaid is not null. I obtain this field from the paths.txt file discussed in chapter 2, and use it to decide whether a student already has an entry in the database (and needs to be updated), or if a new entry needs to be created. Since this mechanism is crucial for the system to work properly, the field is required. Chapter 4 discusses this issue in more detail.

3.2 Users

The script processplan.pl is the only one in the whole system that uses the main read-write account. In other words, it is the only one that needs to be able to input data into the database. Clearly, a malicious user could cause serious damage to the database if he or she obtained the password to this account. To prevent this, I have changed the permission on the file processplan.pl so that it is only readable by the owner. An even better solution would be to encrypt the file, but this approach is too time-consuming during development.

PHP scripts, on the other hand, have to be readable by people other than the owner (since they receive requests from the web). While the person using the browser will never be able to retrieve the password because the PHP script will never output HTML code containing it, users with an account on the Computer Science server could certainly access it. To ensure that no harm comes to the database from this hypothetical threat, I created a read-only account. In the worst case, the malicious user would be able to view the contents of the database, but would not be able to modify it.

Chapter 4: System Input

Two similar scripts, silentplan.pl and processplan.pl, process the personal information files. Silentplan.pl outputs no success or error messages when run: it is silent. Because of this, it is used by the cronjob discussed in chapter 2. Processplan.pl is exactly the same as silentplan.pl, except it prints error and success messages to the screen. It exists for two reasons: first, it allows users to update their .plan files manually, without having to wait for the cronjob to automatically do it for them; second, the messages help users determine the cause of errors. For instance, if a student types "country" as a field name, the script would warn him or her that the field is not recognized. Further inspection would allow him or her to determine that the correct field name is "nationality."

This chapter begins by discussing general formatting rules for .plan files and the reasoning behind them; it then focuses on how the script reads the information in these files and stores it in variables. The chapter ends with a description of necessary adjustments to field values before making insertions into the database.

4.1 Recognized Fields

When a student account is created on the Computer Science server, a .plan file is placed automatically in his or her home directory. An example of this template file follows:

```
------------------------------------------------------
Name:
Office:
Advisor:
Phone:
Fax:     (804) 982-2214
Address: Computer Science Department, Thornton Hall,
Address: University of Virginia, Charlottesville, VA 22903
E-mail:
------------------------------------------------------
```

The fields seen here form the basis for the list of fields that the system recognizes (this list appears in chapter 3). I selected most of the remaining fields by looking at existing .plan files, students' homepages, and by discussion with Professor David Evans of the Computer Science Department of the University of Virginia. For instance, I added the "quote" field because many students include a quotation in their .plan file. I incorporated some fields to make the script more tolerant. For instance, three fields, "name," "last_name," and "first_name," exist. If a student elects to use "name," processplan.pl assumes a comma as the divider between the first and last names. Conversely, the student can choose to fill out the "last_name" and "first_name" fields. Similarly, if a student enters the email field as "email" instead of "e-mail," the system will still recognize it properly.

4.2 Reading the Information

Both processplan.pl and silentplan.pl must be called with the path to the .plan file as a parameter: `perl processplan.pl path=/uf2/fh4u/.plan`. This is because a student's computing id (in this case fh4u) is a required field in the database. If the path does not include three forward slashes, the script exits.

The script handles the text file one line at a time. It begins by determining whether the line is a comment. Following Perl syntax, the script considers any line that starts with a pound sign a comment, and, consequently, ignores it, moving on to the next line in the file. Any text outside of the first two dividers is ignored. In accord with the .plan template, I have defined a divider as any line that begins with dashes.

The script continues by deciding whether the line contains a colon. If it does, the colon is used as a divider between the field name and the field value. Rather than setting the field name to anything that comes before the colon, the script first strips any extraneous tabs, new line characters, and leading and trailing spaces. For example, the system would assume the field name for the line

```
Name                          : Felipe
```

to be "name." Further, processplan.pl makes the field name lower case and converts any spaces in it to underscores, because the field names that the script recognizes as valid (shown in chapter 3) are all in lower case and contain no spaces. Thus, the script identifies "Areas of Research" as the valid field name `areas_of_research`. The script also prohibits "name," "last_name," and "first_name" from having empty values because a student's name is a required field in the database (see chapter 3).

If the line does not contain a colon, the script assumes that the field name is the same as the field name for the previous line, and that the current line's text is the field's value. I added this mechanism because I encountered some .plan files that did not include a field name in each line:

```
office:  (804)982-2391
         Computer Science Department
```

```
                    Thornton Hall
                    University of Virginia
                    Charlottesville, VA  22903-2442
```

Thus, when it processes the second line of this example, the script would assume that the field name is still “office,” and that “Computer Science Department” is the value for the field. If the line contains only spaces, tabs, and new line characters, it is ignored.

Next, the script compares the field name of the line being processed to the list of valid field names. If the field is recognized, processplan.pl stores the field value in the array `@values` and sets a flag in the array `@validFieldFlag`, indicating that the field name has a valid value. For example, the third position in the arrays denotes “advisor”; consequently, if a line contained a value for this field, the script would place it in the third position of `@values`, and would set the third position of `@validFieldFlag` to 1. If the flag for a particular field was already set (meaning that the field already contains a value), the script appends the current value. If a field is not recognized, processplan.pl prints out an error message, while silentplan.pl performs no action.

Since one student can have several email and phone numbers, the values for these fields are stored in separate arrays. The system enforces a limit of five emails and five phone numbers to prevent any student from having an arbitrarily large number of entries in the database.

4.3 Adjusting Field Values

Before the information can be inserted into the database, several adjustments must be made to the values of the fields. First, the script replaces any single quotation marks with two single quotation marks to prevent SQL parsing errors. Next, it ensures that the .plan file contained either the fields “first_name” and “last_name” or the field “name.” If the latter is the case, processplan.pl uses a comma as a divider between the last and first names, in that order. A missing comma causes the script to set the first name to the value of the field and to leave the last name empty. If a student includes all three of these fields, “first_name” and “last_name” take priority over “name.” Finally, if none of these exists, the script prints an error message to the screen and quits.

The database requires the field “sex” to have values “M” or “F.” In order to make the system more tolerant, the script accepts the values “male,” “female,” “f,” or “m,” and converts them to “M” or “F” (the accepted values are case insensitive). The database requires the format of the date of birth to be yyyy-mm-dd. The script accepts the formats mm/dd/yyyy, mm-dd-yyyy, mm/dd/yy, and mm-dd-yy. In the case of the last two, processplan.pl sets the first two digits of the year to 19 if the value of “yy” is greater than 15, and to 20 otherwise. Clearly, this arbitrary delimiter could fail in the future, so I strongly recommend using one of the forms that contain four digits for the year.

Finally, the script performs a check on the picture by using the shell command `identify`. To guarantee that all home pages have a uniform look, processplan.pl requires the size of the picture in pixels to be within the bounds of the following variables:

```
$pictureHeightMax=350;
$pictureHeightMin=310;
$pictureWidthMax=250;
$pictureWidthMin=210;
```

At this point, processplan.pl prints a summary of all recognized fields; sample output follows:

```
****************************
valid field: office, with value: n/a
valid field: advisor, with value: David Evans
valid field: major, with value: Computer Science
valid field: last_name, with value: Huici
valid field: first_name, with value:  Felipe
****************************
```

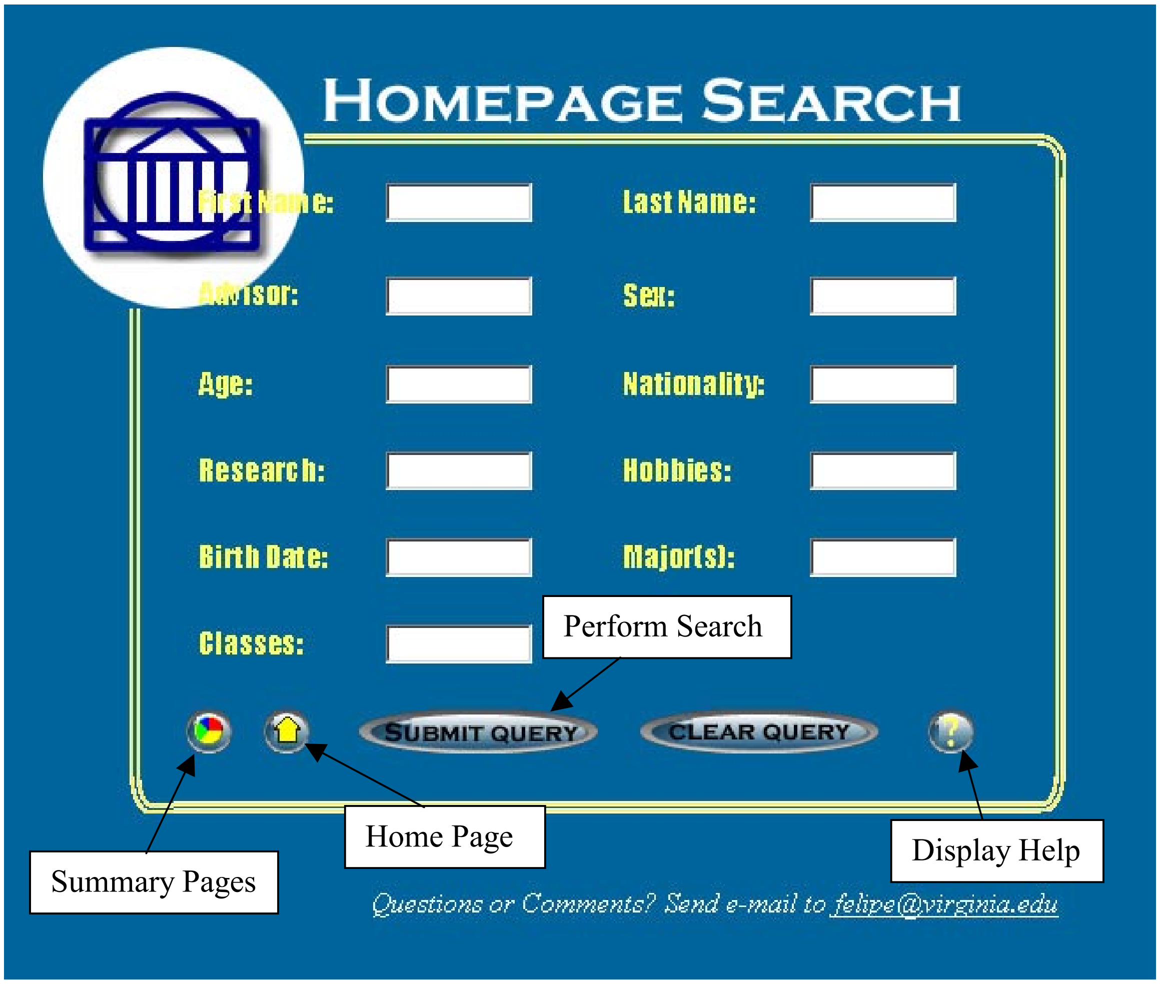
4.4 Input to Database

The script must determine if a record already exists for the student whose .plan file it is processing. If it does, the entry has to be updated; if it does not, the entry has to be created. The system makes this decision by searching the “uvaid” field in the main table of the database. If an entry matches the uvaid of the path to the .plan file, the script knows that it needs to make an update.

This decision does not affect the phones and emails tables. Before inserting the emails and phones in the .plan file, the script deletes all entries matching the current value of “uvaid” from these tables. This allows for easy editing of those fields: if a student wished to remove any emails or phones from the database, he or she could simply delete those entries from the .plan file and run the script again.

Chapter 5: System Output

Any person with Internet access can view, search, and create summaries of the information on the database through the page index.html, located at http://www.cs.virginia.edu/~fh4u/index.html.

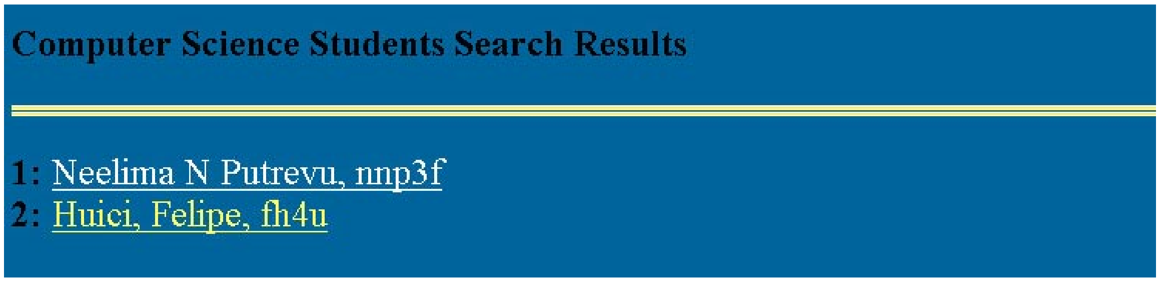


5.1 Search Engine

The screen shot shows that the interface of the search engine resides on the page index.html. Users can search most but not all of the fields in the database. For instance, it would not make sense to search by phone number or email, so I did not include those fields. Users fill out as many fields as they wish, press the “Clear Query” button to clear all the fields, and hit the “Submit Query” button to perform the search. Leaving all fields blank and hitting this button will cause the system to display all entries in the database.

After the query is submitted, a JavaScript function in index.html carries out a few checks before performing the search. First, it ensures that the user has supplied a properly formatted date of birth. The only accepted format is mm-dd-yyyy. The function then checks that the “sex” field contains “M,” “F,” “FEMALE,” or “MALE.” The last check is for the “age” field: it should contain only numbers. If any of these checks fails, the function displays an error message and cancels the query so that the user can fix the mistakes.

If all checks are satisfied, the function submits the query to search.php3, the PHP script that performs the search and displays the results. This script begins by building the SQL string containing the search. Search.php3 performs pattern matching through the SQL operator LIKE and the % character. The % character matches any sequence of characters, including an empty sequence. The script surrounds each field value that the user typed in the query with % signs to match any entry in the database containing the value. For instance, typing “eli” in the first name field and submitting the query results in the following hits:

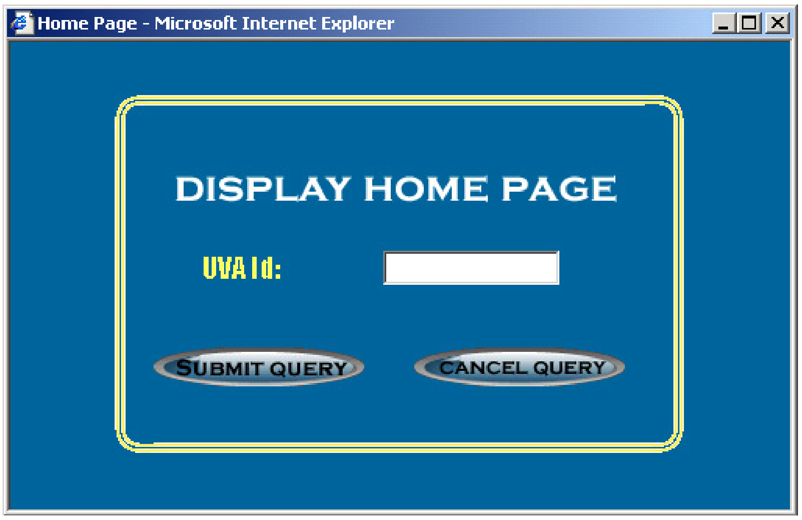

Computer Science Students Search Results

1: Neelima N Putrevu, nnp3f
2: Huici, Felipe, fh4u

Typing “eli” caused the script to match the names “Felipe” and “Neelima.” Note that for each match, the script includes a link to that student’s home page. If there are no matches, the script displays an error message and a link back to index.html.

5.2 Home Page

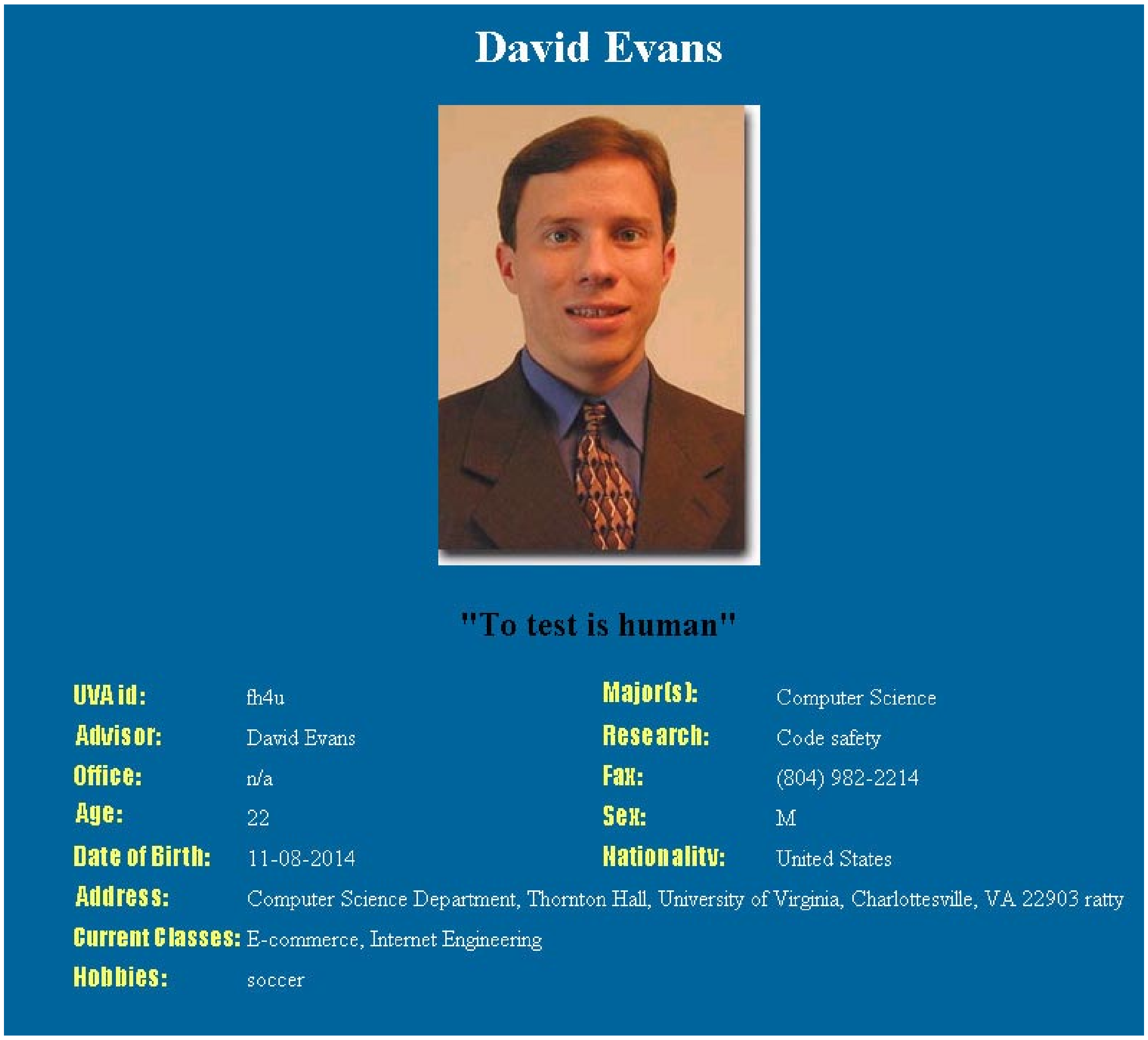
To view a home page, a user can either perform a search and click on the link of one of the matches, or click on the button on index.html labeled “Home Page.” If the user decides to click on the latter, index.html displays the following sub-menu:



I included this option as a shortcut to students who want to view their own pages. By using this form, students can bypass the search engine results page and go directly to their home pages. A simple JavaScript function makes sure that the “UVA id” field contains some text and submits the request; if no page exists with the specified id, homepage.php3 displays an error message with a link back to index.html.

Regardless of which option a user chooses, the script homepage.php3 will receive a request containing the UVA id of the student whose page it has to display. The script uses this id to search the database for a match and retrieve the rest of the fields for that student. Naturally, if no match is found the system will display an error message instead of the home page.

Next, homepage.php3 begins displaying the information row by row. Each row contains at most two fields; if a student has supplied neither of the two fields in the row, the row is not displayed at all. If he or she has only provided one field, this field will get the whole width of the row to itself. I kept long fields such as hobbies, address, and current classes in their own row. Here is a screen capture of a home page that has most of the information filled out (the information is fake):



David Evans

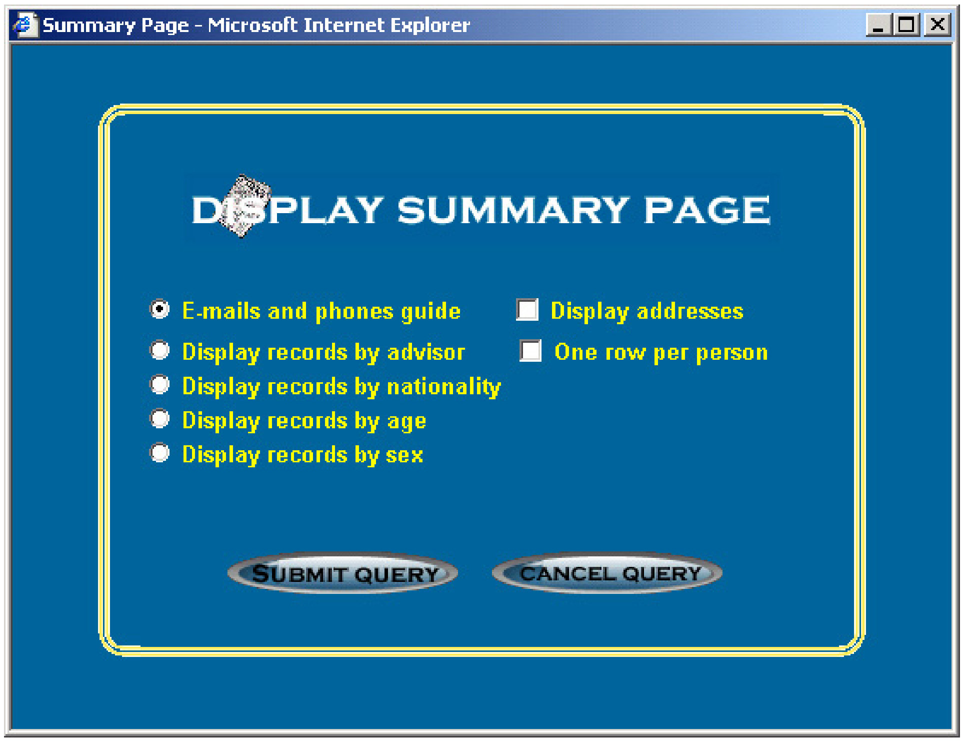
"To test is human"

UVA id:	fh4u	Major(s):	Computer Science
Advisor:	David Evans	Research:	Code safety
Office:	n/a	Fax:	(804) 982-2214
Age:	22	Sex:	M
Date of Birth:	11-08-2014	Nationality:	United States
Address:	Computer Science Department, Thornton Hall, University of Virginia, Charlottesville, VA 22903 ratty		
Current Classes:	E-commerce, Internet Engineering		
Hobbies:	soccer		

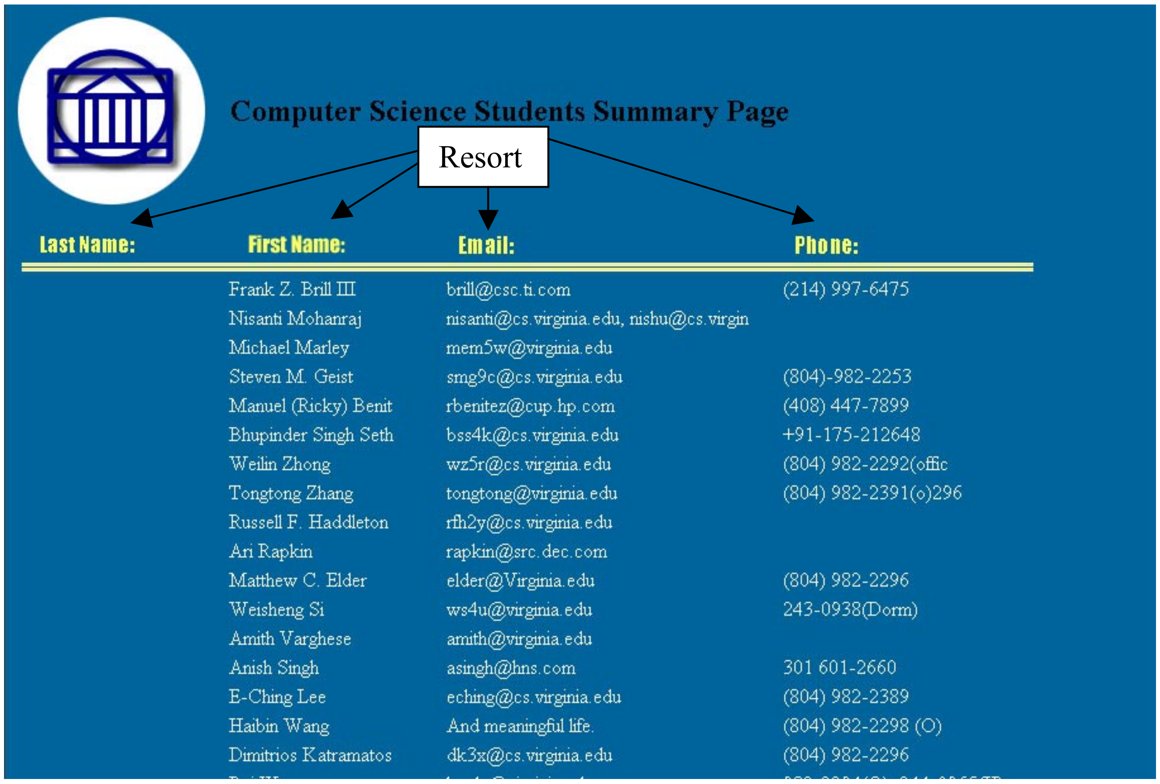
On a final note, the script includes the value of the field "extra_HTML" at the end of the page, but inside the <BODY> tag. This mechanism exists in case a student wants to expand his or her home page beyond this basic template. The system does not verify the correctness of the HTML code within this field, so it is the student's responsibility to make sure that the page looks as expected.

5.3 Summary Page

To display summary pages, a user points the browser to index.html and click on the button labeled "Summary Pages." The system displays the following sub-menu:



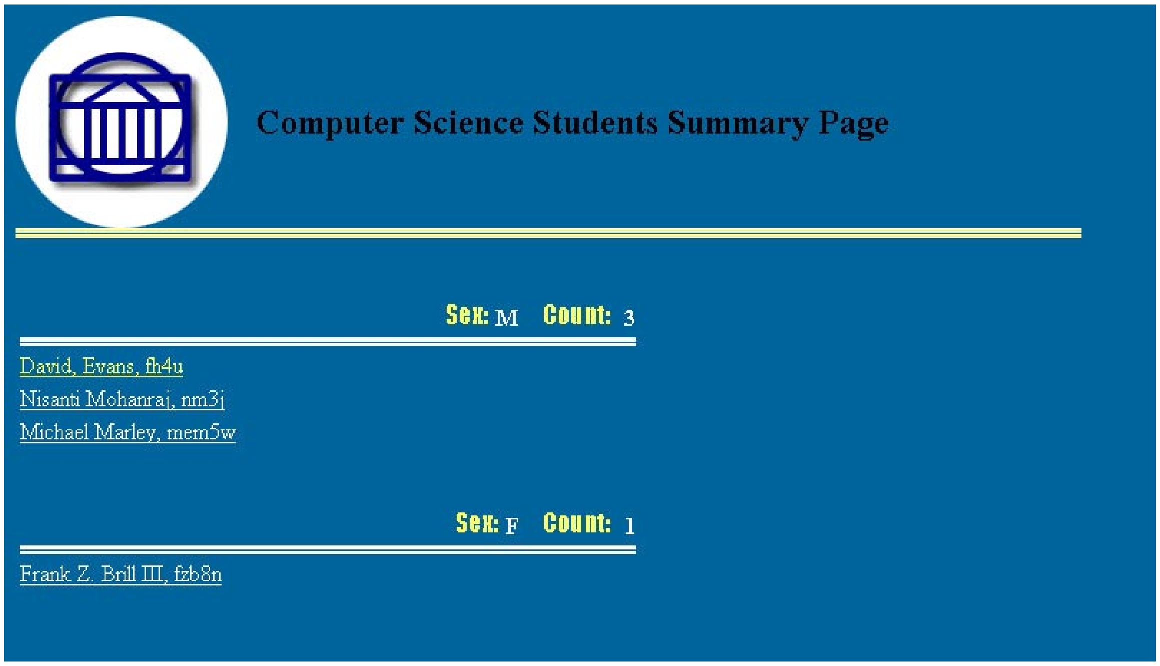
Each one of the radio buttons on the left side represents a different type of summary page. The first option displays a phone directory of all the students in the database. The two check boxes on the right side apply only to this type of summary page. The "one row per person" option causes the system to show only the first phone and email for each person. The following screenshot has only this second option checked:



Last Name:	First Name:	Email:	Phone:
	Frank Z. Brill III	brill@csc.ti.com	(214) 997-6475
	Nisanti Mohanraj	nisanti@cs.virginia.edu, nishu@cs.virgin	
	Michael Marley	mem5w@virginia.edu	
	Steven M. Geist	smg9c@cs.virginia.edu	(804)-982-2253
	Manuel (Ricky) Benit	rbenitez@cup.hp.com	(408) 447-7899
	Bhupinder Singh Seth	bss4k@cs.virginia.edu	+91-175-212648
	Weilin Zhong	wz5r@cs.virginia.edu	(804) 982-2292(offic
	Tongtong Zhang	tongtong@virginia.edu	(804) 982-2391(o)296
	Russell F. Haddleton	rfh2y@cs.virginia.edu	
	Ari Rapkin	rapkin@src.dec.com	
	Matthew C. Elder	elder@Virginia.edu	(804) 982-2296
	Weisheng Si	ws4u@virginia.edu	243-0938(Dorm)
	Amith Varghese	amith@virginia.edu	
	Anish Singh	asingh@hns.com	301 601-2660
	E-Ching Lee	eching@cs.virginia.edu	(804) 982-2389
	Haibin Wang	And meaningful life.	(804) 982-2298 (O)
	Dimitrios Katramatos	dk3x@cs.virginia.edu	(804) 982-2296

The last name field is mostly empty because currently most users do not use a comma as the delimiter between first and last names (see chapter 4). By default, the results are sorted by last name. To display the results in a different order, a user can simply click on the yellow labels for the fields.

The remaining four types of summaries are very similar. Each one of these groups the values for the field into separate categories. For example, in the case of the "advisor" summary page, the script will display all students whose advisor is Andrew Grisham first, those whose advisor is Bill Wulf second, and so forth. For the "sex" summary page only two categories exist, so the script will only have two groups. The script includes a count for each group. The following screen shot clarifies this explanation:



Computer Science Students Summary Page

Sex: M Count: 3

David, Evans, fh4u
Nisanti Mohanraj, nm3j
Michael Marley, mem5w

Sex: F Count: 1

Frank Z. Brill III, fzb8n

Chapter 6: Analysis

This final chapter provides a description of the extent to which the system satisfies the objectives outlined in chapter 1, as well as an analysis of the correctness of the home pages and the search engine. The chapter concludes with a discussion of possible extensions to this project.

6.1 Fulfillment of Objectives

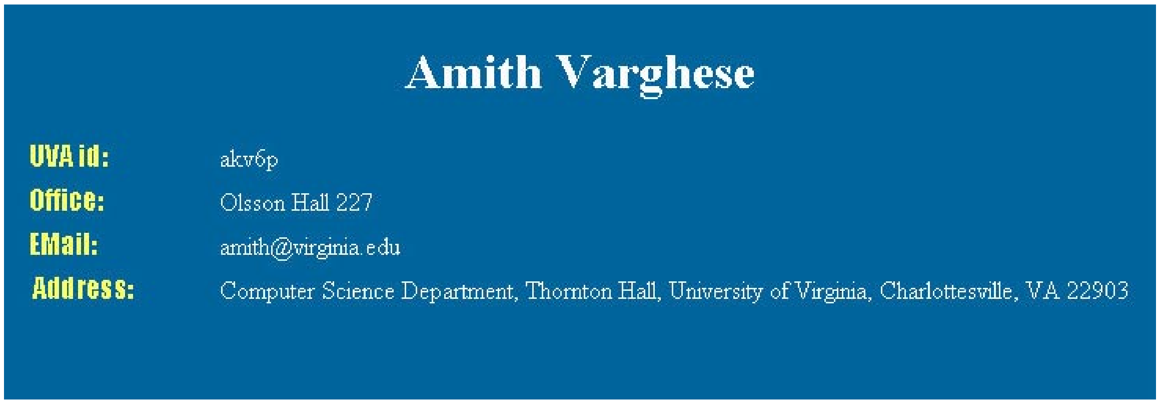
The system I designed achieves the main objective: to reduce effort and save time for its users and to offer a simple means of accessing the contents of personal information files. During testing, I was able to create my own home page within a single minute, a far cry from the time it takes to create one by hand. Users do not even have to wait for the cronjob to enter the data into the database; they can do so manually by running processplan.pl. Because of the system's expeditiousness, I expect that most Computer Science Graduate students who do not yet have a home page will fill their .plan files. With this system, every student in the department should soon have a home page.

Searching the information is also much quicker. If a user wanted to find all students who have a particular professor as their advisor, they could use the search engine in index.html to obtain the results in seconds. Without this mechanism, he or she would be forced to sift through every home directory and .plan file for this information. Such a search could take hours.

While summary pages are not part of the project's main objective, they also save time. Creating a phone guide, finding the number of males and females, and viewing the number of students under each professor takes seconds. However, these summary pages do have a shortcoming. Because their input comes from loosely formatted text files, some of the groupings get duplicated. For instance, inspecting a summary of advisors reveals that James French is listed in two separate groups, one under “Dr. James French,” and the other under “James French.” Thus this professor should have a student count of 2, but each group reports only 1. To prevent this, processplan.pl could cross check the advisor in a .plan file against a list of valid professors.

6.2 Correctness of Information Display in Home Pages

As chapter 5 shows, each row of information in a home page has at most two fields. If both of these fields contain no values, the system should ignore the row; if only one of the fields exists, the script should give it the full width of the row. I viewed over 40 different home pages to check that the system meets these criteria.

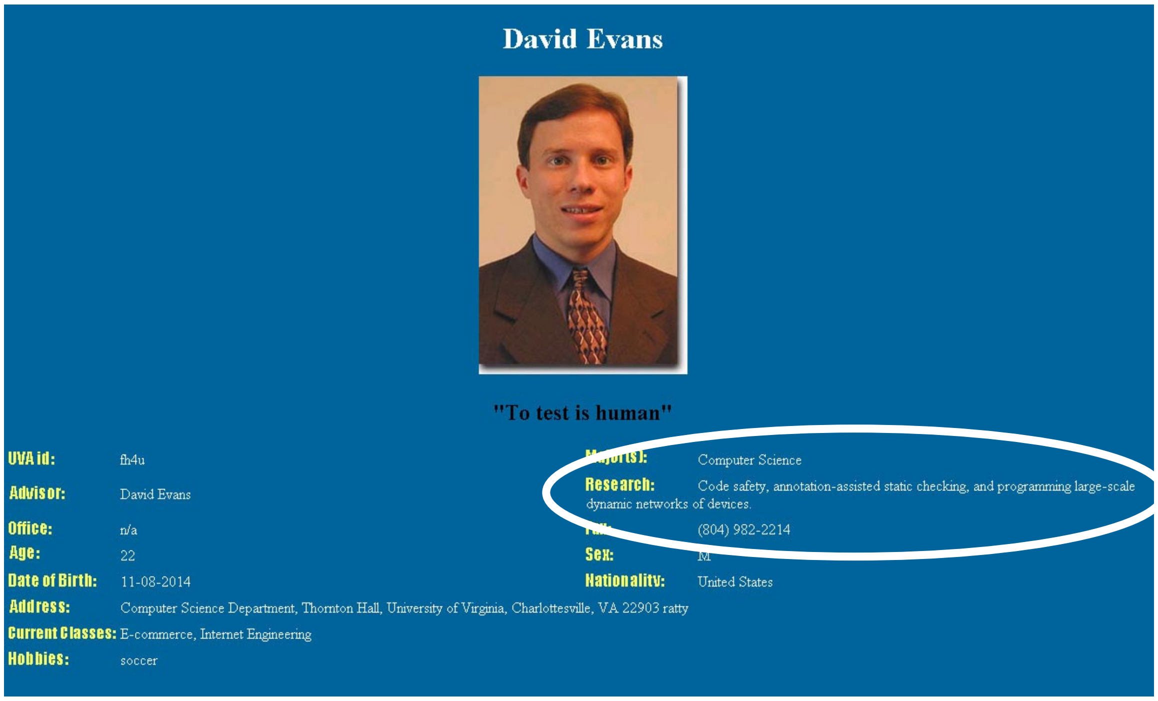

Amith Varghese

UVA id:	akv6p
Office:	Olsson Hall 227
EMail:	amith@virginia.edu
Address:	Computer Science Department, Thornton Hall, University of Virginia, Charlottesville, VA 22903

This student, for instance, has not included any information for the “research” and “advisor” fields. Since these two reside on the same row, the script ignored the whole row. Also note from the screenshot that the student filled out a value for the “office” field but not for the “fax” field. Since these would appear on the same row, the script gives the former the whole width of the row.

I also checked the “date of birth” field for correctness. Since the database stores dates in the format yyyy-mm-dd, I had to make sure that the script properly changed this format into the mm-dd-yyyy format. The home page screenshot in chapter 5 shows that this is indeed the case.

The script contains, however, a few problems. Long field values for fields that share rows can affect the appearance of a page. To illustrate, I created a record in the database with a relatively long value for the “areas of research” field:



David Evans

"To test is human"

UVA id:	fh4u	Major(s):	Computer Science
Advisor:	David Evans	Research:	Code safety, annotation-assisted static checking, and programming large-scale dynamic networks of devices.
Office:	n/a	Fax:	(804) 982-2214
Age:	22	Sex:	M
Date of Birth:	11-08-2014	Nationality:	United States
Address:	Computer Science Department, Thornton Hall, University of Virginia, Charlottesville, VA 22903 ratty		
Current Classes:	E-commerce, Internet Engineering		
Hobbies:	soccer		

Another shortcoming arises from the “extra_HTML” field. The script currently treats the value of this field as HTML code, but the database restricts that size of this field to a few characters. Any HTML code that a user wants to add will almost certainly require more than a few characters, rendering this field useless. To correct this, the value of the field should be an absolute URL to an HTML page containing the code. The script would then be responsible for reading this file and copying its code to the end of the home page.

6.3 Search Engine Correctness

To guarantee the correctness of the results from the search engine, I began by performing a search in each of the eleven fields found in index.html. Once I was satisfied that this worked properly, I submitted queries that used several fields at once, testing cases with only one match, with more than one case, and with no matches. Lastly, I performed searches using only part of a field's value for every field in index.html. For example, instead of typing "Soccer" as the search criterion for a hobby, I only typed "occ." Again, the search engine returned the correct results.

A minor problem with the search engine is that it contains both an "age" field and a "date of birth" field. These are clearly redundant, since no one would search by both age and date of birth. The solution is clear and simple: delete the age field, since it quickly becomes outdated.

6.4 Possible Extensions to the Project

Though the scope of this system is limited to Computer Science graduate students, if it were extended to other groups, they could certainly benefit from it. For instance, an undergraduate student could browse through the profiles of everybody living in his or her dormitory floor, finding common academic and non-academic interests. A student might study better by working with a peer he or she found through the system, or might be more inclined to consistently attend class knowing that some of the people living on the same floor attend it too. He or she might enjoy a more fulfilling college life knowing that people with similar non-academic interests (sports, music, entertainment, etc.) live in the dormitory.

Prospective students could also benefit. Profiles and summary pages of current students will be of use to applicants as they choose a university. Perhaps the applicant is interested in attending a school with a large international community, or one enrolling mostly from the state of Virginia; this system could provide just such information. A prospective student might even contact an enrolled student through e-mail to ask questions that a recruiting booklet cannot answer.

Naturally, this system need not be confined to the University of Virginia. Other schools could certainly implement a similar system and enjoy the same benefits. Moreover, a company could provide this system on its intranet: the summary pages could be used to advertise the demographics of the company, and an employee could search for peers with similar interests.

Even without these extensions, this system has given a powerful method of communication for Computer Science graduate students at the University of Virginia. Students can create a home page, search each other’s personal information, and view summary pages without expending much time or effort.

Works Cited

Dubois, Paul. MySQL. New York: New Riders Publishing, 1999.

Greenspun, Phil. Phillip and Alex's Guide to Web Publishing. San Francisco: Morgan Kaufmann Publishers, 1999.

Kochan, Stephen and Wood, Pattrick. Exploring the UNIX System. New York: Sams Publishing, 1992.

Meloni, Julie C. PHP Essentials. Rocklin: Prima Publishing, 2000.

Perl Mongers, The Perl Advocacy People. Retrieved September 27th, 2000 from the World Wide Web: http://ww.perl.org.

Peterson, Larry and Davie, Bruce. Computer Networks. San Francisco: Morgan Kaufmann Publishers, 1996.

Pierce, Clinton. Sams Teach Yourself Perl in 24 Hours. New York: Sams Publishing, 1999.

Veeraraghavan, Sriranga. Sams Teach Yourself Shell Programming in 24 Hours. New York: Sams Publishing, 1999.

Bibliography

Ashenfelter, John P. Choosing a Database for Your Web Site. New York: John Wiley & Sons, 1999.

Goodman, Danniel. JavaScript Bible. Foster City: IDG Books Worldwide, Inc., 1998.

Kitalong, Karla S. I Hate UNIX. Indianapolis: Que Corporation, 1994.

Muller, Robert J. Database Design for Smarties. San Francisco: Morgan Kaufmann Publishers, 1999.

MySQL AB, MySQL. Retrieved October 14th, 2000 from the World Wide Web: http://ww.mysql.com.

Wyke, Allen R., Gilliam, Jason D., and Ting, Charlton. Pure JavaScript. New York: Sams Publishing, 1999.

Zend Technologies, PHP: Hypertext Processor. Retrieved September 30th, 2000 from the World Wide Web: http://www.php.net.