EPCC-SS-2001-03

The TRACS Web Database

Daniel Mossop

Abstract

In the past tracking terminals allocated to visitors under the EPCC run TRACS programme [1] has proved difficult. The TRACS web database system was started to provide a means of recording the whereabouts of these terminals, and by the start of this project the underlying database had expanded to hold details not just of the machines, but also of the visits. Further growth of the database is likely in the future. This project aims to develop a more general and complete web interface than the one that existed at the start of the project. The system uses a *MySQL* database [4] that is operated on via CGI scripts. The scripts are written in Perl and make use of the Perl Database Interface (DBI) [5].

1. Introduction

1.1. EPCC and TRACS

EPCC was established in 1990 as part of the University of Edinburgh. The Centre was formed to accelerate the uptake of High Performance Computing (HPC) systems throughout academia, industry and commerce.

EPCC has been coordinating the EC-funded TRACS (Training and Research on Advanced Computing Systems) programme since 1993 [1]. TRACS allows EC-resident academic and industrial researchers to come to the EPCC for visits lasting between four to thirteen weeks. During this time they can use EPCC resources to help with their research.

As part of the programme the visitor is given access to EPCC-owned terminals, which are distributed to host departments, mainly located in Edinburgh, for the duration of the visit. It is important that the locations of these terminals are recorded in a clear and consistent manner. In order to facilitate this an on-line database accessible through the web is being developed which aims to aid in tracking this hardware.

By the start of the Summer Scholarship Programme (SSP) [2] the range of information able to be stored in the database had already increased beyond tracking the terminals. This new information included details of the visits, the visitors, their hosts and their contacts. Further information may need to be added in the future, so the design of the database and web front-end should be kept as general as possible. Their implementations should also be kept independent of one another, making the front-end portable to different database systems.

The aim of this project was to complete the existing implementation of the web interface and generalising the implementation to facilitate future expansion. In addition, there were a number of suggested improvements that could be made if time permitted, such as implementing a system to allow data to be inserted to the database from an XML file (applications to TRACS are made via the web, and the entered data is stored in an XML file).

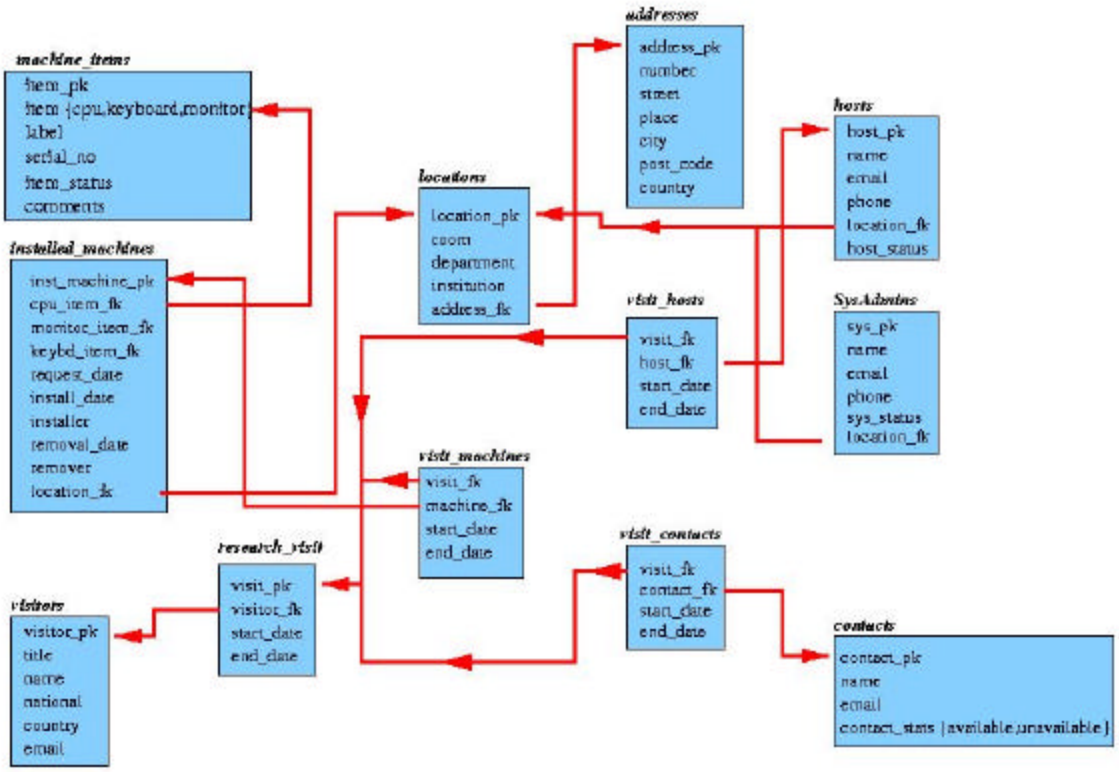
2. Initial System

A partial implementation of the TRACS system was already in place at the start of the SSP. This is as a result of the work done by Mario Antonioletti, Elson Mourao and Elena Breitmoser since winter 2000/01, when the project was started. Their implementation consisted of two main parts, a database and a web interface.

2.1.The Database

Based on a design by Elson and extended by Mario and Elena, the database was in the form shown in figure 1, at the beginning of the SSP.

Figure 1: The Initial TRACS Database.



In this form the database is able to store a range of data relevant to the TRACS programme. The original purpose of the web database (to keep track of the location of EPCC terminals) is dealt with by four tables in the above diagram: *machine_items*, *installed_machines*, *locations* and *addresses*. The terminals consist of three items, a CPU, a monitor and a keyboard. Each of these items makes up a single entry in the *machine_items* table. The table holds details of the items, such as their serial number, their status (ie. whether they are in need of repair or not) and what the item is (CPU, monitor or keyboard).

The *installed_machines* table links to the *machine_items* table for three of its fields. These links specify which CPU, monitor and keyboard are part of the terminal. The rest of the *installed_machines* table holds details about its installation, e.g. who installed it and when. It also links to the *location*s table, allowing the location of the terminal to be recorded. Each time a terminal is installed somewhere a new *installed_machines* record will be created, without removing details of its previous installations. This will allow the history of the terminals to be retained. The *location*s table specifies a room within an institution (for instance, a university) and links to the *addresses* table which specifies the city, street and number of the institution.
The database shown has expanded to include data other than details of the terminals. The *visitors* table allows details of TRACS visitors' name, nationality and email address to be stored. Likewise the *contacts*, *hosts*, and *SysAdmins* tables allow details of the contacts and hosts (local researchers collaborating with the visitor) and EPCCs' systems administrators (who are responsible for the distributed terminals) to be stored.

On their own, however, these tables would be of limited use. The usefulness of the database comes from the fact that the *research_visits, visit_machines, visit_contacts* and *visit_hosts* tables allow records in the previous tables to be associated with one another. In this way, details of a TRACS visit can be built up, by associating a visitor with the visit(s) and then assigning (one or several) machines, contacts and hosts to it.

The database in this form allows questions about the TRACS visits (such as: where is terminal X? or, who is the contact for visitor Y?) to be answered simply.

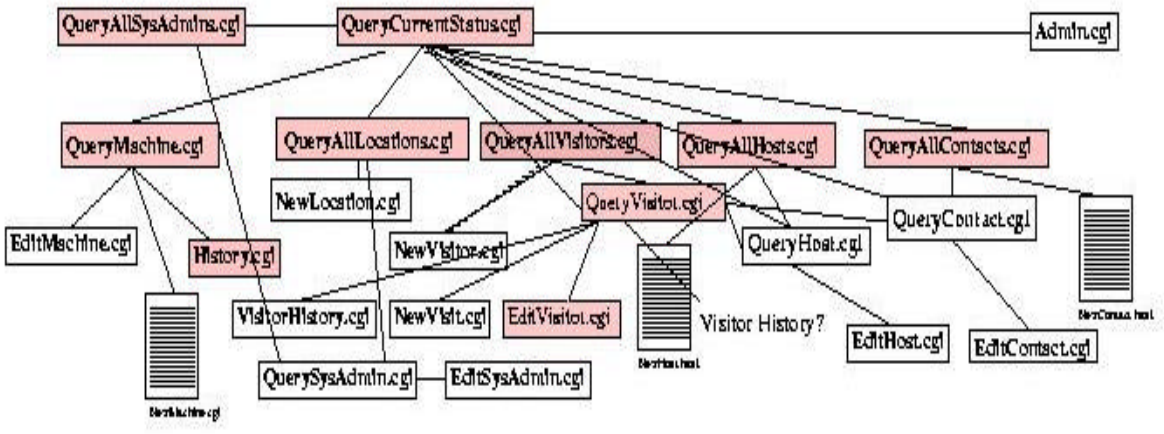
The database is implemented in *MySQL*, a robust database product that implements a subset of the *SQL* relational database programming language.

2.2.The Web Interface

While the above questions could be posed directly to *MySQL* on the command line, it is not really a practical solution, since the queries may at times be quite complex and the users may have limited knowledge of *SQL*. A better solution is to create an interface that allows the user to pose a question easily, perhaps by selecting menu options or pressing buttons, and then generates the corresponding database query. The query can then be passed to MySQL and the returned results displayed in a sensibly formatted manner. Mario and Elena began the implementation of such an interface. The interface is web based, making it accessible to distributed users. It is written in Perl and uses the Perl Database Interface (DBI) and the Common Gateway Interface (CGI). DBI provides a common method of connecting to databases, which not only allows it to connect to *MySQL*, but also allows it to connect to many other databases products without the need to alter the database accessing code. CGI allows the web server to dynamically generate HTML pages, and through its use an interactive interface can be created that can be displayed on most Internet browsers.

The figure 2 shows most of the Perl scripts that had been implemented by the start of the SSP (the shaded boxes indicate completed scripts, the white boxes indicate partially complete scripts).

Figure 2: The Initial Implementation Scripts.



There are four main categories of script featured in the figure 2. These scripts are the *QueryAll* scripts, the *Query* scripts, the *Edit* scripts and the *New* scripts. The *QueryAll* scripts displayed all records of a given type (visitor, host, etc.) in an HTML table. The script for displaying all machine records also adds colour-coding to the records in the table to indicate whether they were associated with past, present or future visits. The *Query* scripts displayed a single record of a given type. These scripts sometimes displayed more information about the records than was displayed by the Q*ueryAll* scripts (which provided a summary only). The *Edit* scripts allowed an individual record of a given type to be edited, and the *New* scripts allowed an individual record of given type to be created.

There were also other scripts at the start of the SSP, other than those shown on the diagram. Some of these supported the *Query* and *QueryAll* scripts by providing routines to retrieve the relevant records from the database. Others supported the *Edit* and *New* scripts by providing routines for writing data to the database. See Appendix A for the full list of scripts that were implemented by the start of the SSP.

3. Objectives

This project aimed to implement the TRACS web database system to the point that it was sufficiently functional to allow it to be put into use. There were a number of areas of development that needed to be addressed to achieve this.

3.1. The Database

There is some information relevant to the TRACS visits that can not be accommodated by the database as it was at the start of the programme. This includes data such as the IP addresses of the terminals, details of the visitors' arrivals and accommodation and diary records of meetings between the contacts and visitors. The database should be extended to handle this.

3.2. A General HTML Table Generator

One requirement of the Web Database is that it should have a method of displaying summaries of the information contained in the database. These summaries originally implemented as HTML tables by the *QueryAll* scripts mentioned in the section on the initial system. This involved having a script for each table that could be displayed. If this could be replaced by a single routine that could generate all the required tables then the amount of time and new code necessary to handle expansions to the system might be reduced.

3.3. The General Record Manipulator

As well as being able to view summaries of the data, the user also needs to be able to work with individual entries, viewing them and editing them. The user also needs to be able to create new entries. This functionality was originally partially implemented by the *Query*, *Edit* and *New* scripts. A general routine that could replace these scripts could significantly increase the ease with which database expansions could be handled.

3.4. Web Database Views

When the system is in full use there will be several classes of users. Some will be allowed to view records only, while some will be allowed to create new records or alter existing ones. In order to accommodate this an access system that gives different user classes different views of the database could be implemented.

3.5. The Administrative Layer

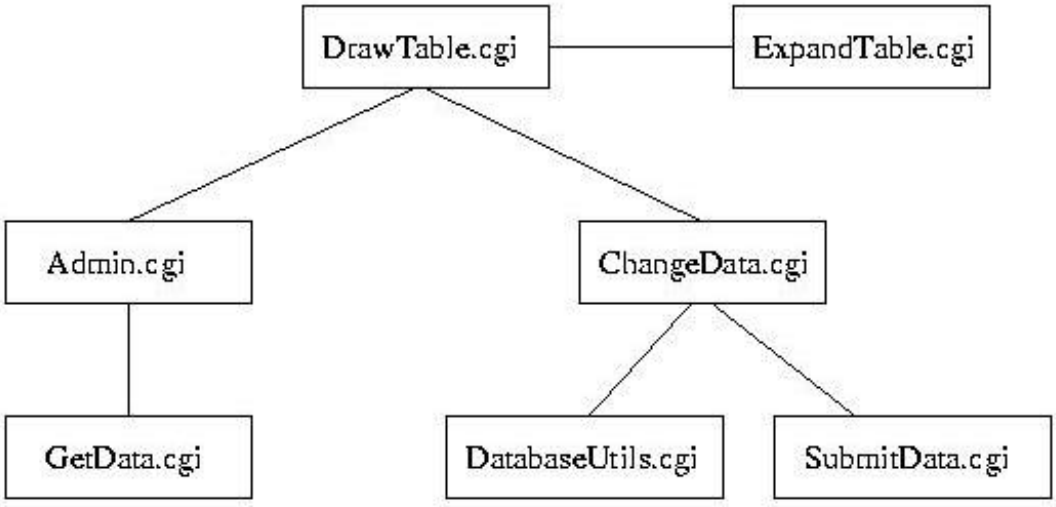
There may be times when it would be desirable for a system administrator or similar to directly alter the contents of the database. While this is possible by accessing the *MySQL* database directly, an HTML front-end would provide a much simpler interface. The Administrative Layer would also provide the only way of removing records from the database. Such a database view would necessarily have a restricted access.

4. Final System

This section will examine the extent to which the project objectives have been achieved during the course of the SSP. It will examine each of them in turn, before finishing with a look at issues that remain outstanding.

There have been major changes to the files that implement the system during the course of the SSP. Figure 3 shows the main files that are now used, and how they interrelate.

Figure 3: The Current TRACS Scripts

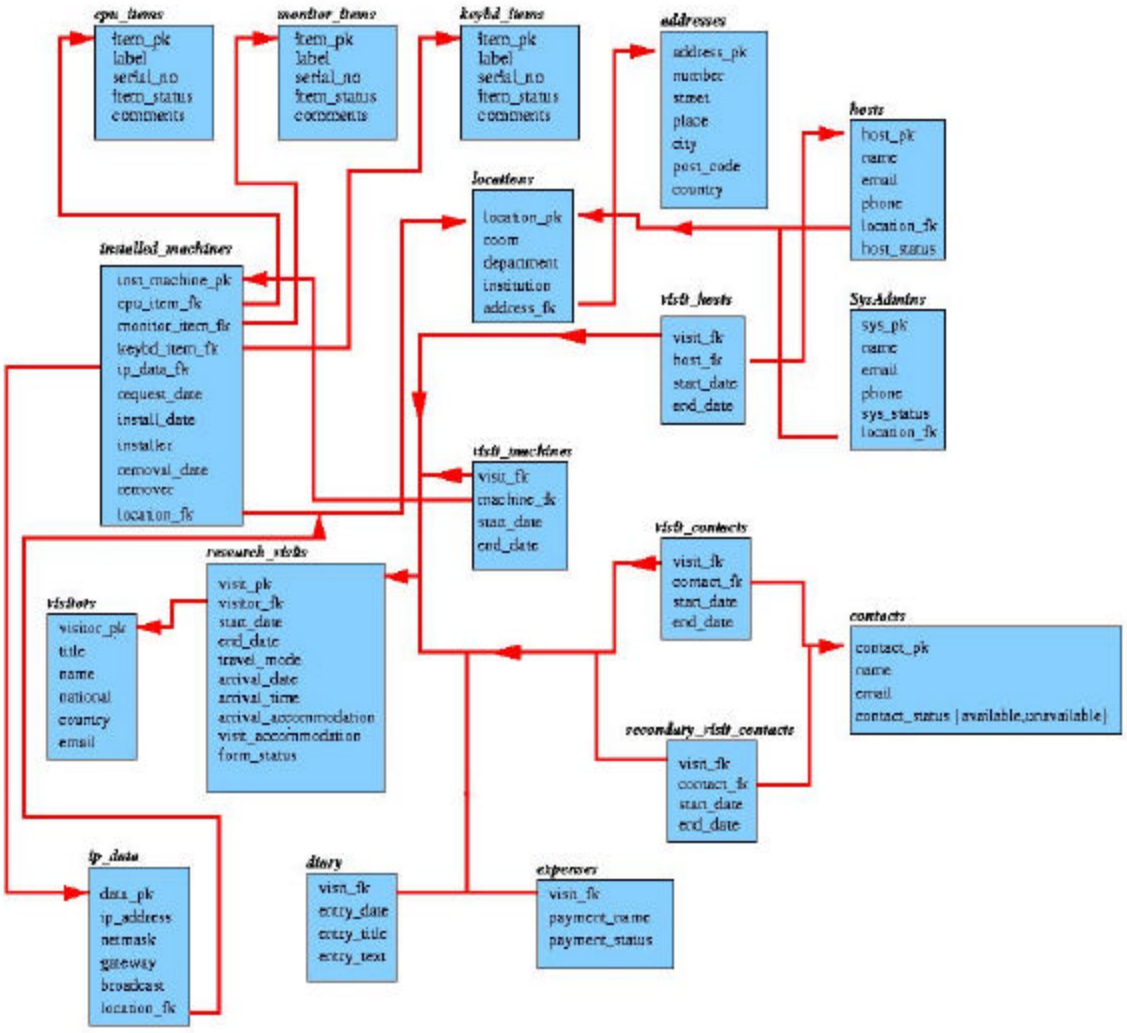


DrawTable.cgi contains the routine that generates the HTML tables. The routine calls the SQL query builder located in *ExpandTable.cgi*. From *DrawTable.cgi* it is possible to link to the administrative layer implemented by *Admin.cgi* with support from *GetData.cgi.* It is also possible to link to the individual record manipulator. This is implemented by *ChangeData.cgi* which makes use of routines in *DatabaseUtils.cgi* and *SubmitData.cgi.* For a full list of the files now in use, see Appendix B.

4.1. The Database

The database has been altered during the course of the SSP for two main reasons. One is that changes to the database have allowed the implementation of the Web Interface to be simplified. The other is that extensions to the database have made it possible to store more TRACS data than before. As a result of these changes, the database appears as shown in figure 4.

Figure 4: The Current TRACS database.



One of the first changes made to the database was to separate the *machine_items* table from the initial database into three separate tables, one for each type of machine item: CPU, monitor and keyboard (*cpu_items*, *monitor_items* and *keybd_items* respectively). This adds to the number of tables in the database, but reduces the amount of data that must be stored since there is now no need for an explicit field indicating the item type. Previously the *installed_machines* table was joined to the *machine_items* table on three of its columns. Now it has the property that it is joined to other tables on at most one of its columns (e.g. *installed_machines* is joined to *cpu_items* by one column only – *cpu_item_fk*). This property is now true for all tables in the database. This simplification helped to keep the amount and complexity of code required for a general table routine (discussed in the next section) to a minimum. This was the only change necessary to allow the completion of the implementation that had been started.

All other changes to the structure of the database have been made to extend the range of TRACS data that can be stored. The *ip_data* table has been added to allow the IP addresses (and other

related information) of terminals to be stored. This is linked to a new field in the *installed_machines* table. The initial database was only able to specify one contact per visitor, while in practice visitors can be assigned both a primary and secondary contact. This is now addressed by the *secondary_visit_contacts* table. The *secondary_visit_contacts* table creates associations in the same way the original *visit_contacts* table does, the table holds the keys of the visit and contacts that are to be associated and specifies a start and end date for the association. The *research_visits* table has been extended to include details about their arrival in Edinburgh and their accommodation during the visit. During the visits TRACS visitors receive periodic payments (subsistence, travel, etc.). This is now held in the *expenses* table, which links to the *research_visits* table (ie. Each visit has associated expense records). Another table that links to the *research_visits* table is *diary*. The *diary* table allows the TRACS contacts and hosts to record the outcome of meetings with the visitors, and permits entries to be very large if necessary (up to 4Gb).

4.2. The General HTML Table Routine

One of the most important functions the Web Database must be able to perform is to summarise selected data from the database. As happened in the initial implementation, the data is displayed in HTML formatted tables. The aim of this section of the project was to produce a general routine robust enough to be able to create tables from any given combination of columns from the database.

On first inspection the database appeared to be not at all suited to a general routine. The layout did not seem to be significantly regular to be described by a single routine. Changes to the database were considered, in order to try and increase the regularity. Only one alteration was made to this effect. The *machine_items* table was split into three tables: *cpu_items*, *monitor_items* and *keybd_items*, as mentioned in the last section. Where previously there had been three links from the *installed_machines* table to the *machine_items* table, there was now only one to each of the three new tables. The condition that any two tables were linked on at most one entry was now true for the whole database. This allowed the implementation to be simplified significantly. Since this condition is assumed by the general table routine, it must be preserved by any future expansion of the database.

The situation was also complicated by the tables that link between two other tables (ie. *visit_machines*, *visit_contacts* and *visit_hosts*). Because they link between other tables they do not have their own primary keys (they are uniquely defined by the composition of several columns). This would mean that extra code would be required to handle these tables. However, the entries in these tables only ever relate to a single TRACS visit. It seemed unnecessary to have a table displaying all of the entries, when they are largely unrelated to each other. It was decided, then, not to attempt to make the general table routine capable of handling these tables, but instead leave them to be displayed by the general routine for manipulating individual entries.

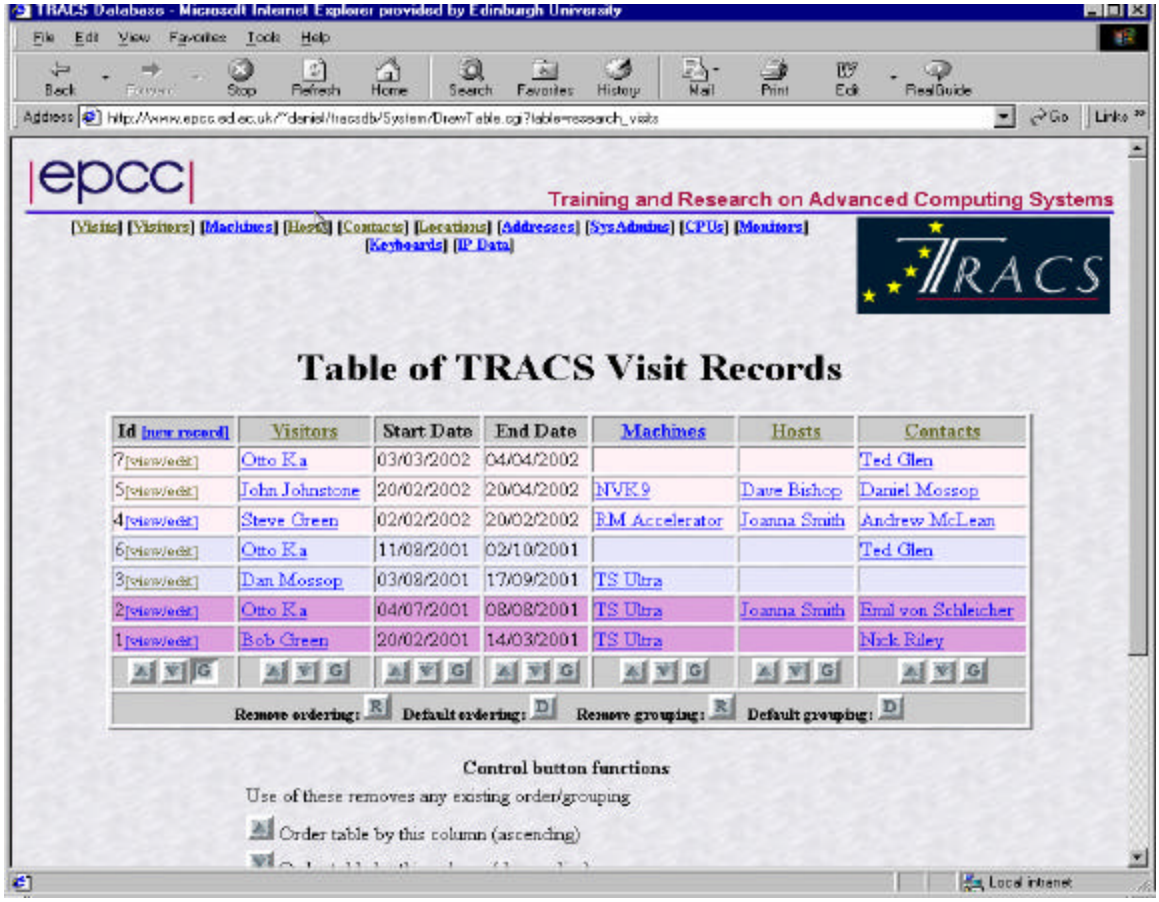
This view of the database was now sufficiently simple to allow the creation of a general table routine (which is in the file *DrawTable.cgi*). The first thing that had to be done was to declare information about the structure of the database. This included specifying, in a definitions file (in *Definitions.cgi*), the joins between tables. The SQL language actually permits these joins to be declared when the tables are defined, but this feature is not implemented in the version of MySQL currently in use for the database. Other details of the database were also written to the definitions file. Some, like the names of the database tables and columns can actually be found out directly from the database which would save the need for new declarations to be added when the database is expanded, though currently this is not implemented. Others, like the display names

for the columns and tables and the aliases for the tables are not available from the database, and must be specified[1]. These are used to make the Web Database more informative to the user. A further list of definitions was needed to declare, for each table that the routine would be used to create, the names and locations of the columns to place in the table and the specifics of any ordering and grouping on that table. This list also specifies the path from the table holding the first column, to the tables holding the other columns of required data.

Since any HTML table the routine creates must be populated with data from the database, it was necessary to write a general routine that produces SQL queries to extract the required data. By using the path lists for the tables defined in the definitions file, the SQL query builder (found in *ExpandTable.cgi*) is able to traverse the database joining together all the tables necessary to allow all the relevant columns to be selected.

Once all the data has been obtained from the database, it is relatively simple to display it in a HTML table. A sample output table is shown in figure 5.

Figure 5: A Table Generated By DrawTable.cgi.



This example not only illustrates the way in which the data has been compiled, it also highlights some of the features available to the user while working with the data. The buttons at the bottom of the table allow the ordering and grouping that is applied to be altered. The table rows are

[1] the aliases specify a list of columns from the table which provide enough information to allow the user to distinguish individual entries from one another

shaded according to whether they are future entries (the top three rows), current entries (the next two rows) or past entries (the last two rows).

There are a number of links from the table to other pages. The row of links at the top of the page (starting with *[Visits][Visitors]…*) allow all the other tables generated by the routine to which the user has access (access issues are discussed in the *Web Database Views* section) to be reached. The same function is performed by the links from the table headings, except for the first column. All the links in the body of the table, again excluding the first column, link to a view of the record associated with that entry. The reason this is a view only is that it avoids the situation where the user might be want to change, for instance the contact associated with the visit, but would instead actually replace the existing contact record by mistake.

The first column is the only place to provide links to editable records (and these are replaced by links to the non-editable versions if the user does not have the proper access). The links in the body of the first column allow the user to edit the record in that row. Further information, not displayed in the table can also be reached this way. The link at the header of column one allows a new record of the type being displayed to be created.

4.3. General Record Manipulator

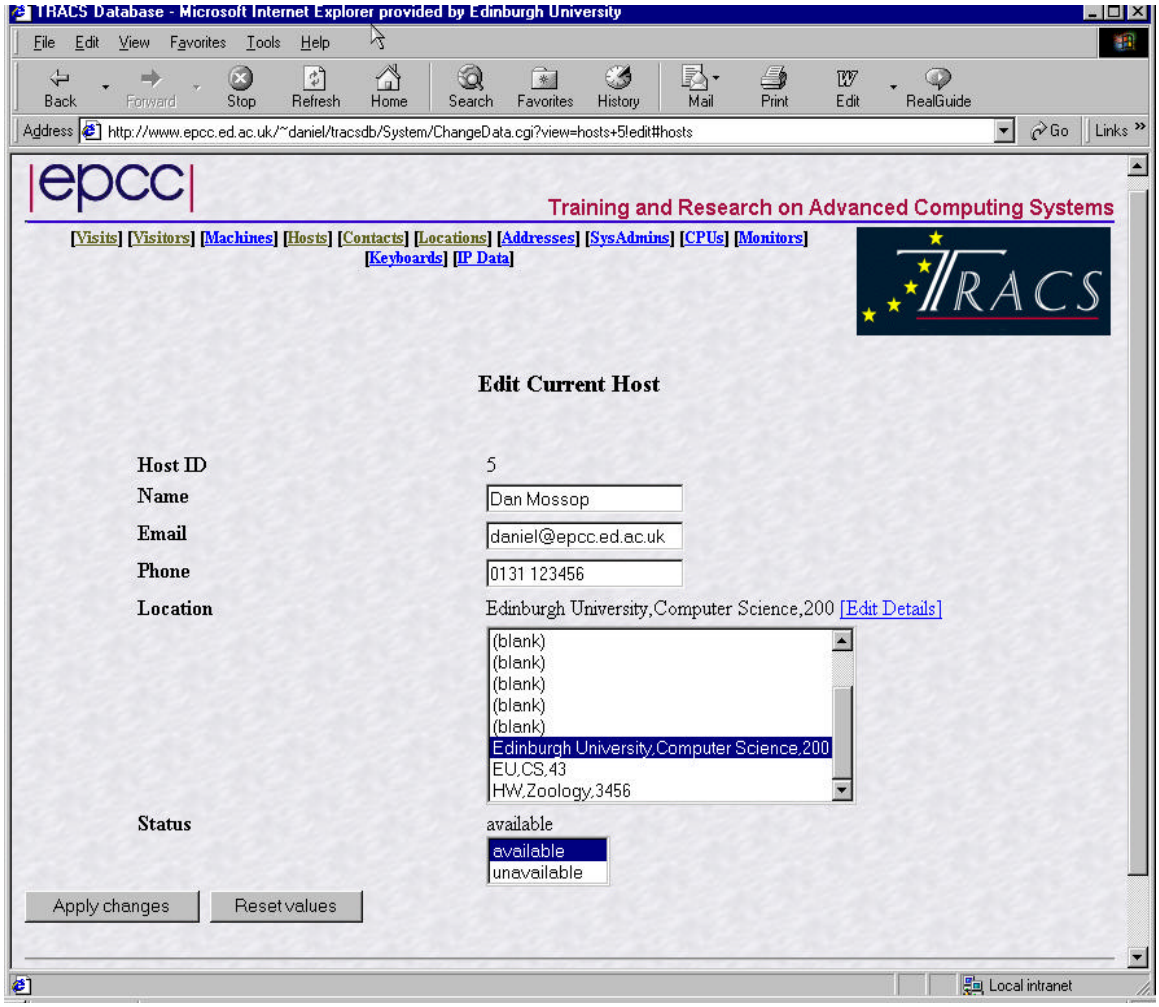
Another very important feature of the Web Database is its ability to allow users to view, edit and create individual database records. In the current implementation this functionality is controlled by a general routine allowing it to be largely independent of the actual application of the database.

Of all the issues involved in the design of the general record manipulator, the most difficult was how to structure it so that entire visit records could be built up in a logical manner. It would be a trivial matter to update a single table entry, such as the information about one host. It would not be so trivial, however, to organise it so that a visit could be declared and associated with a visitor, a contact, a host and a machine, each of which must also be declared and perhaps even associated with locations and addresses. Because of the way the database tables were linked together, normally in one-to-many relationships, it would not be an easy matter to update more than one database table at a time. So it seemed that each table must be treated individually, but this went against the idea that the entire records (e.g. a visit and all its associations) should be declarable in one go.

One solution to this was to make the content of the HTML page alter dynamically to reflect the current circumstances. For instance the host associated with a visit could be displayed on the same page as the location for that host, and the address for the location. Then if the host was changed, the location and address could be changed at the same time. However, to implement this would require the use of a client-side language such as Javascript. Such browsers are often poorly supported across different platforms and browsers, which can lead to unreliable results when the code is run on browsers other than the ones on which it was developed. In order to avoid this, the output from the Perl scripts was limited to HTML, which is supported well across browsers.

The solution that was finally implemented (by *ChangeData.cgi)* was a variation of this, which can be implemented using server-side page generation only. The page would not alter dynamically, instead a single record would be displayed in either a viewable format or editable format as shown in figure 6.

Figure 6: An Editable Record Generated by ChangeData.cgi.



The user can then reach other records by following the links on this page (for instance, in the above diagram the [Edit Details] link will allow the selected location record to be edited). When the records are accessed by these links, the original record is displayed at the top of the new page in a read-only format.

The mechanism for handling new entries works in the same way as that for handling editing and viewing. When a new entry is created it has blank entries that can then be edited as normal.
At each stage the changes must be applied to the database using the provided button, or the data will be lost.

Because the general record manipulation routine only allows you to move to linked records, all the records that affect the current one, and no others can be reached. This helps to ensure that the creation and editing of records is logical.

Some of the tables in the database (the ones with no primary key) do not have HTML tables created by the general HTML table generating routine discussed in the last section. As a result these records can not be edited by linking from such a table. The way they are handled is that they appear in the individual record entry for the table to which they link (in the case of the current

implementation it is always in the visit record entries). The associations between the current record and others, as defined by these tables, appear in a list. Each of these entries can be edited and new entries created by following the links provided.

4.4. Web Database Views

When the system is running, the Web Database will have several classes of users. Examples of these classes and their requirements are:

- Administration – Must be able to access the entire database and the administration layer (discussed in the next section).

- Systems – Must be able to view, edit and create data relating to the EPCC terminals and their installation. Should not be able to alter visitor records, etc.

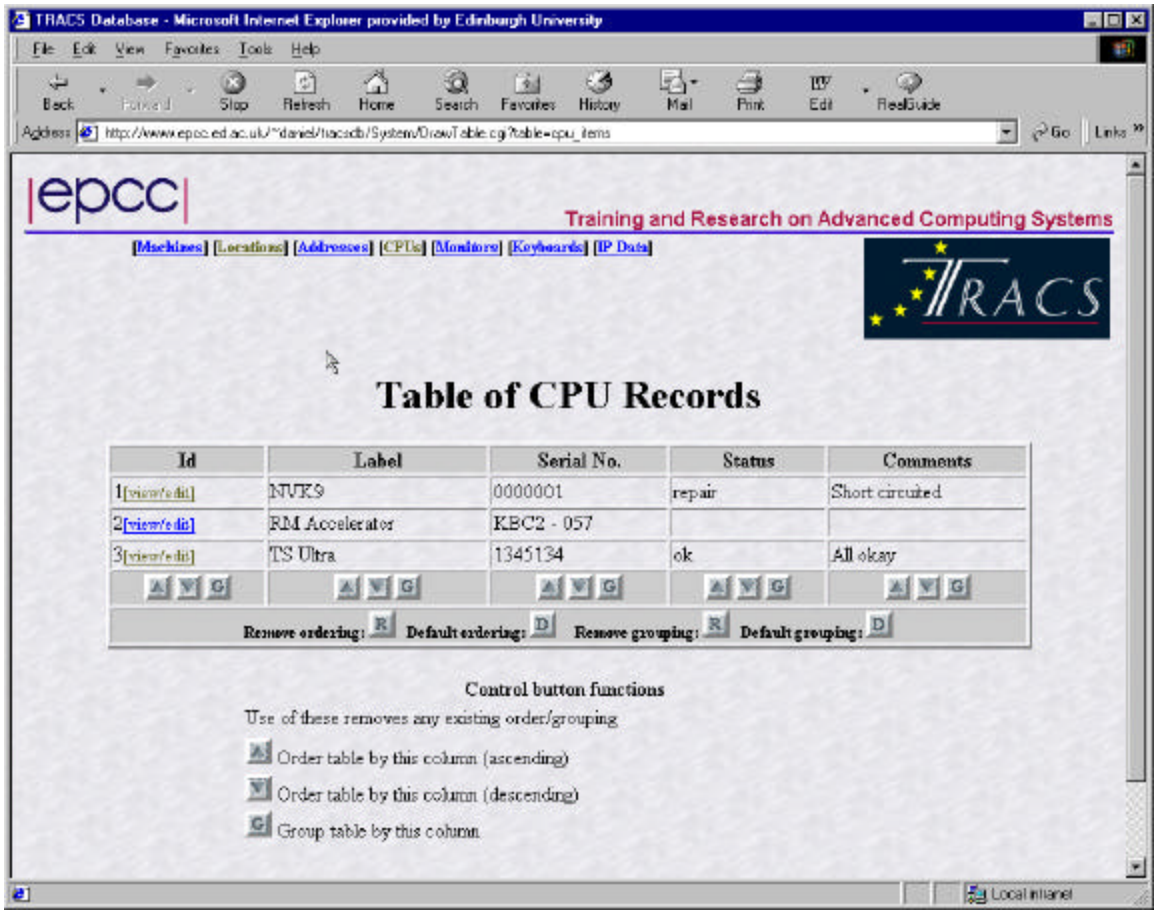
- EPCC-Staff – Should be able to view all the records, but not be able to make changes to the data.

This functionality is implemented by the *Access.cgi* file. For each user class it declares a list of tables that can be accessed and whether that access permits editing or not. It also has a function that can be called from any of the scripts and returns the access level for a given table. This allows access to each table to be individually controlled.

In order to implement this it is necessary to have some way of deciding which class the user belongs to. This is achieved by using the htaccess password protection facility [5]. When the users attempt to access the site they are prompted for a username and password. If they enter a valid password they gain access to the site, otherwise they do not. When they have access to the site an environmental variable is set that contains the name of the user class. This allows the correct view of the site to be displayed for the user.

Figure 7 shows a page created for the systems class. It can be seen that there are no links to tables such as the one holding the visitors' data. Attempts to access these tables result in an error page being displayed.

Figure 7: An Page Generated using the Systems Login.

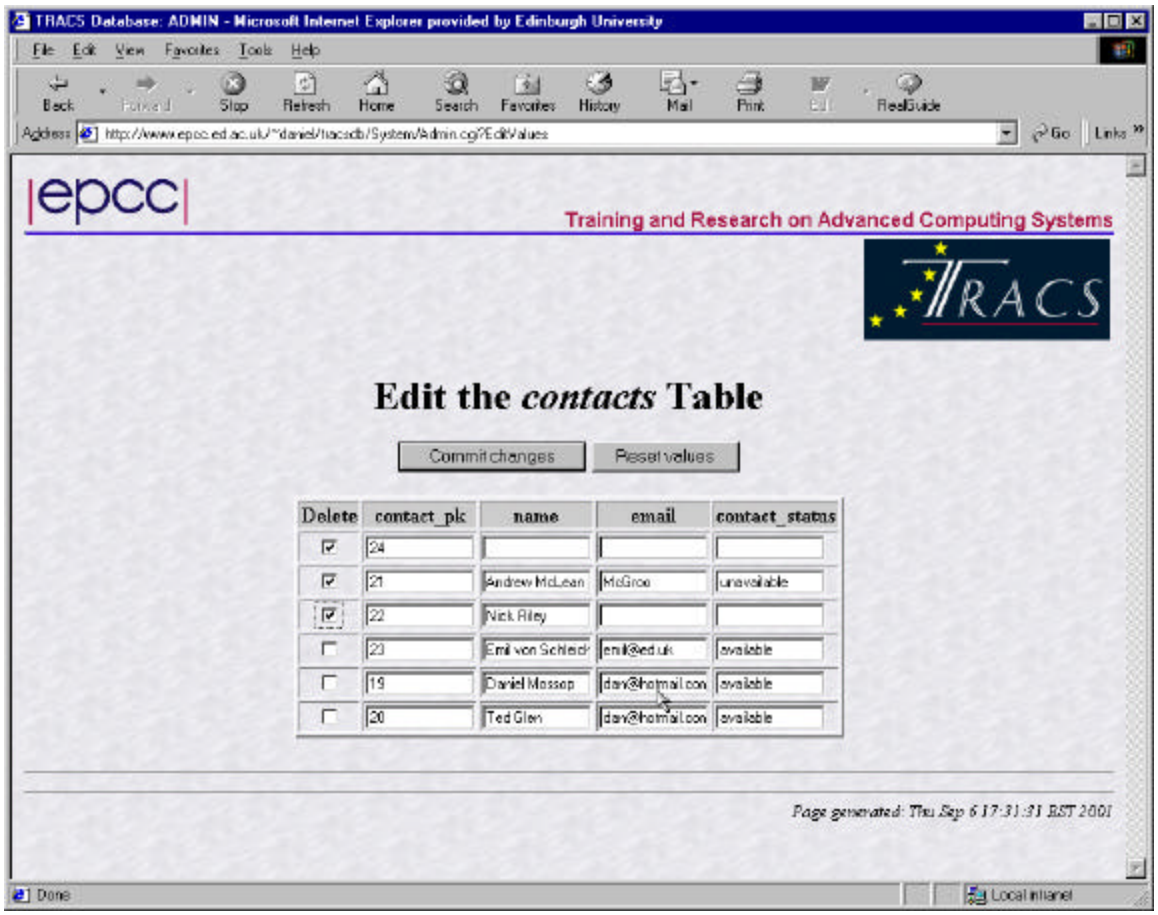


4.5. The Administrative Layer

As it stands at the moment the combination of the general HTML table generator and the general record manipulating routine provides a method of viewing and editing all the data in the database, as well as allowing new data to be added. It does not, however, allow data to be deleted. While this prevents users from accidentally deleting records, it might at times be useful to remove entries, if they have been entered twice by mistake, for instance. For this reason an administrative layer has been created that allows direct editing of the database tables, but uses an HTML front-end to avoid the need to enter SQL commands directly. The administrative layer was implemented in read-only form in the initial system, that is, it allowed the contents of the database tables to be viewed, but not edited.

During the SSP this has been expanded to include editing of entries and the deletion of rows. There is a simple interface, with all the columns being represented as text boxes, and with each row having a checkbox, allowing it to be marked for deletion. Figure 8 shows the main page of the interface (there is also a page that asks for confirmation of the changes).

Figure 8: An Example of the Administration Layer.



4.6. Outstanding Issues

At the end of the SSP some issues remain with respect to the implementation of the Web Database. The system as it currently stands is close to being fully operational. There are some points to take into account however.

Passing some non-alphanumeric characters between pages as part of the URL argument causes an incorrect page to be displayed. This is caused because the characters (such as < and >) are not properly escaped before they are passed. Currently the URLs are encoded and decoded by handwritten routines that can not handle all the necessary characters. These routines should be replaced by the escape and unescape functions in the Perl CGI module.

The administration layer is fully functional, in that it allows data in the database to be manipulated directly (as detailed in section 4.5). There is a problem with the access restrictions, however. At the point where the table to be displayed is selected from a pull down menu everything seems to be working. But when the table name is submitted and a new page displayed (the one that should display the table), some thing happens to remove the *$ENV{REMOTE_USER}* environmental variable. Since this variable stores the login name of the current user, after this point it is no longer possible to verify that the user has access to the administration layer. This must be resolved if the administration layer is to be included in the final system.

Currently the username and password used to access the database are hard-coded into the scripts. Before the site is put into operation a more secure method of retaining the values throughout the session should be maintained.

There are a few other alterations that could be made to the Web Database, that are not important (in so far as they do not seriously affect the functionality) but would improve the user interface.

Currently all textboxes for data entry are a standard size, apart from the diary and date entry boxes. These boxes are not necessarily large enough to properly display the full entry (although it can still be entered). It would improve the display if the boxes matched the maximum size of the entry.

Another feature that would improve the system would be the validation of the input before it is applied to the database. This would allow, for instance, incorrect dates to be trapped and the user informed.

Similarly it would be desirable if errors generated during database access were trapped and reported.

5. Conclusion

The Web Database was started with a view to tracking EPCC terminals, but has since grown to encompass a significantly larger set of data. This project has aimed to develop an interface capable of handling not only the current requirements of the system, but also future expansion of the database. It has attempted to resolve this by the creation of general routines that are not fixed to a specific database design.

There are some issues that still need to be addressed before the system is fully operational. These are detailed in section 4.6. Despite this the system is sufficiently operable to demonstrate the degree to which the needs of the TRACS programme have been realised.

One of the biggest proofs that the general routines are indeed robust enough to allow database expansion, and are able to make this process fast came when the database had new tables added. This happened near the end of the project when four new tables were appended to the database. In adding the tables, only the definitions file needed to be edited, not any actual code. The only exception to this was the diary that required the new feature of using HTML textareas to handle the potentially large entries. This feature is now also available to be used in future expansion. The process of adding these tables was fast and simple, a vast improvement on having to create a new script for each table.

The system that has been implemented during the course of the programme is not a perfect system and will certainly be subject to alterations in the future. It does, however, provide a solid base for these alterations. It is sufficiently functional to allow it to be used with after only the fairly minor changes detailed in 4.6. The flexibility of the system means that it should be easily adaptable to handle changes to the underlying database, or even to be used for entirely different databases.

Appendix A

This appendix gives details of the perl scripts that implemented the TRACS web database at the start of the Summer Scholarship Programme.

Query scripts

These scripts displayed a suitably formatted HTML table showing the data contained within a subset of the columns in the database tables. These scripts were further divisible into ones that displayed all the rows in the examined columns and ones that displayed a single row entry.

Of these scripts the following were completed to the point of providing the basic functionality required for the system:

```
QueryCurrentStatus.cgi
QueryAllLocations.cgi
QueryAllVisitors.cgi
QueryAllHosts.cgi
QueryAllContacts.cgi
QueryAllSysAdmins.cgi
QueryMachine.cgi
QueryVisitor.cgi
QueryLocation.cgi
QuerySysAdmin.cgi
History.cgi
VisitorHistory.cgi
```

Still in need of work in order to achieve basic functionality were:

```
QueryContact.cgi
QueryHost.cgi
```

Edit scripts

These scripts allowed alterations to be made to records already in the database. They consisted of HTML forms, whose entries already contained the current data, which allowed data to be edited.

The following edit scripts offered basic functionality:

```
EditVisitor.cgi
EditSysAdmin.cgi
EditMachine.cgi
```

Work was still needed to achieve it with:

```
EditContact.cgi
EditHost.cgi
```

New scripts

These scripts allowed new records to be created. These were based on HTML forms with blank entries into which the new data could be entered.

The scripts that offered this functionality were:

```
NewVisit.cgi
NewVisitor.cgi
NewHost.html
NewContact.html
NewLocation.cgi
NewMachine.cgi
```

The following script was not yet at this level of functionality:

```
NewSysAdmin.html
```

In addition to these three categories, there were other scripts used by the system. A summary of them is given here.

GetData.cgi

This acted as an intermediate between the Query and Edit scripts and the database. Each subroutine in GetData generated an SQL query and posed it to the database. The resulting data was then returned to the calling function (located in a Query or Edit script). GetData.cgi offered sufficient functionality to support the calling functions.

AddData.cgi

In a similar fashion to GetData, AddData acts as an intermediate between the database and scripts that write data to it. Each subroutine in AddData generates an SQL statement to write data passed from the calling function to the appropriate database tables.

AddData.cgi was only partially complete at the start of the SSP. It supported the following functionality:

Adding visitor records
Adding contact records
Adding machine records
Updating visitor records
Updating system administrator records

Other functionality required by the system included:

Adding host records
Adding location records
Adding visit records
Adding system administrator records
Updating contact records
Updating machine records
Updating host records
Updating visit records
Updating location records

Admin.cgi

Admin.cgi allowed the user (typically an administrator) the ability to view the contents of the database tables as HTML pages. This was read only at the start of the SSP, though it was hoped that it would be extended to allow the database to be edited from these pages.

MiscUtils.cgi

Contained a collection of routines called by other functions, for instance generic HTML headers and footers.

global.ph

global.ph was included by other scripts, because it contained a definition of the physical location of the scripts and because it included the GetData.cgi and MiscUtils.cgi scripts

index.html

This page just redirected the user to the main page (i.e. QueryCurrentStatus.cgi).

Appendix B

This appendix details the files in the TRACS Web Database system at the end of the SSP.

Access.cgi

This file declares the tables that each user class can access. It also provides a routine that allows other files to determine the level of access the user has on a given table.

Admin.cgi

This file implements the entire administration layer. It consists of six stages:
1. Allow selection of a table from a list.
2. Display the table
3. Edit the table
4. Confirm the changes
5. Apply the changes to the database

ChangeData.cgi

This file implements the single record manipulation routine.

DatabaseUtils.cgi

This file provides routines for retrieving data from the database.

Definitions.cgi

This file contains definitions detailing the layout of the database and the format of the tables to be displayed by the general HTML table routine.

DrawTable.cgi

This file implements the general HTML table routine.

Error.cgi

If an access error occurs, this file is called. It displays an error message and then redirects the user to the intro page.

ExpandTable.cgi

This contains the routine for building general SQL queries. It is called from DrawTable.cgi

GetData.cgi

This contains database accessing routines used by Admin.cgi

Intro.cgi

This file displays an introduction to the interface.

MiscUtils.cgi

This declares routines to generate the page headers.

SubmitData.cgi

This file declares routines for entering data into the database. Called by ChangeData.cgi

Images

This directory contains the graphics used by the interface.

Index.html

This page just redirects the user to the intro page.

References

[1] The TRACS Homepage
www.epcc.ed.ac.uk/tracs

[2] The EPCC Summer Scholarship Programme Home Page
www.epcc.ed.ac.uk/ssp

[3] Perl in a Nutshell – A Desktop Quick Reference
Ellen Siever, Steven Spainhour and Nathan Patwardhan
O'Reilly 1999
ISBN : 1-56592-286-7

[4] MySQL and mSQL
Randy Jay Yarger, George Reese and Tim King
O'Reilly 1999
ISBN : 1-56592-434-7

[5] Managing Internet Information Services
Cricket Liu, Jerry Peek, Russ Jones, Bryan Buus and Adrian Nye
O'Reilly 1994
ISBN : 1-56592-062-7

Biography

My name is Daniel Mossop.

I am 21 years old and come from Dumfries, in the South-West of Scotland.

I am about to start my fourth and final year here at the University of Edinburgh, studying Computer Science BSc (Hons).

I enjoy hillwalking, running and cycling.

I would like to thank my supervisors, Mario Antonioletti and Elena Breitmoser, for their tireless input and advice.