Tying Everything Together with a Relational Database

by Pat Hildebrand *

Rapid change in delivery method means that the information which is necessary to access data is also changing rapidly. The density at which a tape or cartridge was written is critical information for reading the data on it while if the data is online the location is critical. On way of handling such changes in required information is to store it in a relational database.

What exactly is meant by a relational database? Fully relational databases, satisfying all conditions set out in Codd's definition, may only exist in theory and would be more than needed for this discussion. Here what is required is first that there be some degree of normalization of the data. As an example some studies have only one dataset while others have many, a record that tried to anticipate how many datasets is not likely to be very practical so put the study information in a study record and the dataset information in a dataset record with one record for each dataset and a key to the study record. Then it is necessary that there be some way of accessing these records together so that it looks as if two (or more) separate records are really one. Structured Query Language (SQL) is the accepted (complete with an ISO standard) way of doing this for a relational database.

Using the example of studies and datasets in a simplified version, let's say we have two tables which is the accepted term in relation databases for the structure in which the records, called rows, are stored. The first table is called studies and has fields, columns in relational terms, study_num and title and the second table is called datasets and has the columns study_num, dataset_num, and name. Now a very simple example to make sure we are all on the same page. You would like to have a list of all study titles and the names of the datasets for each study. You could issue the SQL command

```
select studies.title,
datasets.name
   from studies, titles
          where studies.study_num
= datasets.study_num;
```

This will give you a list of study titles and dataset names which the title repeated for each dataset. Since you did not request study_num or dataset_num you will not get them back.

The experiences being talked about here used Relational Database Management Systems. These applications were started under Ingres and migrated to Oracle when the university wide choice of a database system made the switch. When dealing just with data some of our researchers have used SQL under SAS. The focus here is not on the specific relational database but rather on the concepts so specifics should be taken as concrete examples rather than the only way of doing it.

Although querying the database directly using SQL is an option and it is possible to use SQL scripts in place of some of the things used here what will be talked about are Oracle Forms applications (developed with Oracle Developer/ 2000), perl scripts, and C programs. The C could probably be replaced with perl but it was a very ambitious undertaking written by the system administrator. When a perl script has to interact with the database it is currently oraperl which is perl4. The perl5 scripts use DBI/ DBD::Oracle and will go into production when the C application has been successful tested out against a more recent version of Oracle.

Because they are probably only of interest to show the wide range of uses I will briefly touch on the UNIX administration applications. The C application (using Pro*C precompiler to interface with Oracle) is a print accounting program that keeps track of printing on our UNIX cluster and our NT network. Users are given an allotment for printing each semester and must pay for additional printing. At the end of each semester a reconciliation is done from a cron job to zero out any allocated funds not used and put in the next semester's allotment. The cron job is a perl script. The application for providing additional funds (user paying, refund because of bad printing, etc.) is a Forms application.

The other administrative application is for keeping track of our users. Since not everyone on campus can have an account on our system this application must check with another database on campus to determine if a person is in the correct department and has the correct status (no undergraduates). This is accomplished by means of a

database link. I don't know terminology for other databases but with Oracle a database link is a means of accessing a table in another Oracle database as if it were a table in the local database. The printing is related to system users by a column indicating what allocation of print funds they get.

Data access is a bigger issue as while we provide computing to a limited group we provide data to the entire university. Since we require the use without an account on our system to show up in person and present then campus ID we originally developed applications under Ingres that were used on our UNIX system in character mode so that calling in from home did not present a problem. When the applications migrated to Oracle and developed under Developer/2000, we found that it didn't make sense to develop in character mode any longer. Even with PPP when people called in from home they were still using character mode as the campus software had vt220 emulation. The database had become even more a part of the application under UNIX as that is when we started putting data on line so that there was even more information that we were keeping track of although the user probably used less of it.

When we designed the database we had been using tapes and cartridges for obtaining the data and at first the data was still used on the mainframe where the access was via a tape job. The tables included one for tape labels which also kept track of the tapes that were used for backing up accounts, temporary use, blank tapes entered into the system but not yet used, etc. Another table contained tape information such as the density, the character set, and whether the tape was labeled. At first glance it might appear that these two tables are each one row per table but the labels table has text which could require more than one row. Other tables are for the individual files on the tape.

When we started putting data online we could redesign the tables dealing specifically with the tapes to include online information or use the fact that the database was relational and use an additional table for the online information. We choose the later since not all data was being placed online.

Data requests, information about what is online, and the library system for hard copy documentation all share tables about the studies as well as having their own tables. For data are current system is a mixture of perl scripts, Forms applications, and even some cgi scripts for web access to the information. The web access takes care of the university wide access to the information. People on campus but outside those who have accounts on our system still have to present an ID and request access in person but now they know if the data are on campus. If the data are on campus they are able to get some information such as the size of files that they are talking about before they make the request so that they can make sure they have the space. We still get people who only want a few numbers, often a statistic that they would have to calculate from a very large dataset, thinking that everything will fit on a floppy that already has a number of files on it, but the web seems to have found users who are better prepared to make use of the data when they come over to request access.

The heart of our data system is a series of perl scripts. When new data are received information is entered into the database about the study and dataset numbers and the type of file (data, documentation, program, etc.). This is actually a Forms application. As the information is entered a todo file is written with information from this table and a number for later identifying the file internally. Please note that there is additional information entered form a master-detail relation in the form and programmatically.

Perl scripts to read tapes/cartridges or process files which have been ftped or are on other media such as CD-ROM use the todo files to determine if a file should be read and do the processing. The todo files are also generated for existing tape/cartridge files to be put online and moving some files but not all from a CD-ROM to disk so the issue of whether or not a file should be read is real. When processing is complete (some, unfortunately, is still manual checking) a file is written with information that should go into the database as well as information needed to move the file from the processing area to where it is accessible to the users.

The names of these files are placed in another file that is read by a nightly cron job. The actual moving of the data and recording of the information in the database is done by this script. Some of the other things done by the script are to check the type of file, find out from the database where that type of file should be moved, check that there is enough space for the file, if need be and there is one available set the database to use a new directory, check that there is not currently a file in the directory with that name, and send e-mail about problems.

The library application is able to tell whether we have any hard copy documentation for a specific study and wether it is on hand or checked out, check thing out, and check them back in. If someone already has a copy of a specific piece of documentation checked out they are not permitted to check out a second copy of the same thing. Also if someone has overdue documentation checked out the application will say what is overdue and no further check outs are permitted for that individual until the overdue documentation is returned and the situation cleared is some other way.

All of the user information for requesting data and checking out documentation is the same table so that

changes do not have to be entered in multiple locations. As much of the information as possible is look up information from other tables. This not only makes the entry simplified but it also makes for fewer errors' in addition to avoiding typos this avoids ambiguous entries such as "student".

There are a lot of relations that exist in the services that we provide. Using a database allows us to restrict who can do what at what time. Using a relational database for the necessary information has made for greater accuracy, a simplification of things since once we have entered an individual say for data we don't have to do turn around and enter them again for checking out the documentation, and the ability to do automate some of the work.

References:

Date, C.J. with Hugh Darwen. *A Guide to the SQL Standard*, 3rd Ed. Reading, MA: Addision-Wesley Publishing Co. 1993.

Edelstein, Stephen. *Learning Oracle Forms 4.5: A Tutorial for Forms Designers.* New York, NY: Relational Business Systems. 1995.

Gundavaram, Shishir. *CGI Programming on the World Wide Web*. Sebastopol, CA: O'Reilly & Associates, Inc. 1996.

Musciano, Chuck & Bill Kennedy. *HTML The Definitive Guide.* Sebastopol, CA: O'Reilly & Associates, Inc. 1996.

Wall, Larry, Tom Christainsen & Randal L. Schwartz. *Programming Perl*, 2nd Ed. Sebastopol, CA: O'Reilly & Associates, Inc. 1996.

http://www.hermetica.com/technologia/DBI

Oracle Developer/2000 Documentation

Forms Developer's Guide
Release 4.5
Part No. A32505-1

Forms Reference Manual, Volume 1
Release 4.5
Part No. A32509-1

Forms Reference Manual, Volume 2
Release 4.5
Part No. A32510-1

Forms Advanced Techniques
Release 4.5
Part No. A32506-1

Forms Messages and Codes
Release 4.5
Part No. A32508-1

* Paper presented at IASSIST/IFDO '97, Odense, Denmark, May 6-9,1997. Pat Hildebrand, Social Science Computing, University of Pennsylvania, pat@ssc.upenn.edu