This paper was presented at CUMREC '97, The College and University Information Services Conference. It is the intellectual property of the author(s). Permission to print out copies of this paper is granted provided that the copies are not made or distributed for commercial advantage and that the title and authors of the paper appear on the copies. To copy or disseminate otherwise, or to republish in any form, print or electronic, requires written permission from the authors.
Abstract

Most of us are planning to update or replace our legacy applications. Some of us have even started. All who have thought about it know that it will take years to conclude a conversion on all of their systems. When you accept the fact that new vendor products will not be ready for at least a year or two the timetable can look depressing. Meanwhile, there are bills to pay, transcripts to produce, payrolls to meet, grants to track, and fundraisers to host. And now we must do it faster, for less money, satisfy more users, with slicker interfaces, and fewer people - and get started on the conversion.

At the University of Missouri we are mixing one-part data warehouse, one-part Web, and one-part legacy/transactional system to produce a faster, cheaper, slicker, more widely-used hybrid that will satisfy our customers while allowing us to convert the underlying systems to client/server. This presentation will describe the implementation details of our work including: warehousing legacy data, reporting from the warehouse via the Web, and capturing and feeding transactions from the Web, through the warehouse and back to the legacy systems.
The Warehouse, the Web
and the Legacy Database

Introduction

At the University of Missouri we believe that the replacement of our batch-oriented, terminal-based, mainframe database systems is inevitable. The exact nature of the replacement technology is tough to pin down but most of us agree that it will possess a GUI interface and will have many characteristics associated with distributed client-server technology. All of us know that it will take a lot of time and enormous effort to make the transition.

While we wait for the vendors to deliver usable products there are a number of steps we can take to prepare for the migration. This paper will describe the things the University of Missouri is doing to prepare for the replacement of our core administrative systems. These activities include building data warehouses for ad hoc reporting, constructing Web interfaces to the warehouse, and using Web forms to capture transactional data which is then batch-fed to our existing administrative systems.

The payoff for our effort is the opportunity to learn the new technology, a chance to define and refine the process of migrating the existing data to relational environments, the benefits of an easy-to-use reporting system, and the efficiencies achieved from the Web-based systems we are building.

The Warehouse

The University of Missouri is a four campus system with 55,000 students, 20,000 employees and a budget of about $1.2 billion. Our grant, accounting, and human resource systems are common to the four campuses. Our student and fund raising/development systems are based on common code, but the data for each campus is held separately. These applications run on an IBM MVS system in our main data center in Columbia, Missouri.

Our first efforts at building a data warehouse began in 1992 and focused on standardizing a collection of student data for system-wide reporting. This traditional data warehouse stores aggregate data from fixed points in time. It does not contain transactional details and it is updated only at certain census points. No effort is expended to keep the data current on a daily, weekly or even monthly basis. We have accumulated census point data for about 6 years. The system was originally implemented in DB/2 on an RS/6000 but is now housed in Oracle on an HP 9000.

With the ever increasing bang-for-buck in microprocessors, memory, and disk we have been able to build a new kind of warehouse in the past year. While our early initiatives in the student
realm made no effort to capture all of the data, we expended enormous effort to standardize the data. Our current approach attempts to grab everything but we do minimal clean-up. We concluded that buying more disk was cheaper and easier than attempting to discriminate among the data worth keeping. We also learned that for most of the data it was not worth the time and effort to do any serious normalization, standardization or other clean-up.

This second warehouse initiative started with human resource data, followed by financial data, and will soon include a complete copy of our student data. Our grant tracking system has been largely replaced by a client-server system (homegrown Oracle/MS Access) that serves as its own data warehouse. We are in the middle of converting our fund raising/development database to client-server (BSR Advance on Sybase NT). The data warehouse is an important tool in this conversion.

**Human Resource**

Our human resource data is moved to the warehouse on a weekly basis. The transfer is accomplished using Mark IV extracts that produce files that are either appended to tables in the warehouse or simply replace the data from the prior period. The bulk of this process was engineered to load the data into DB/2 on the RS/6000. The limitations of the DB/2 loader made it essential to pre-process the data. We use Mark IV extract programs that mostly existed for other reporting purposes. The extracts are invoked weekly as part of the regularly scheduled maintenance activities. The result files are FTPed to our Unix machine where a Perl script executes preserving all grants, constraints, and indexes; truncates the tables when necessary; runs the loader; and then restores the grants and constraints and rebuilds the indexes.

**Financial**

When we began work on our financial systems we made a switch to Oracle on an HP 9000. Our financial data is held in VSAM datasets. The Oracle loader is able to handle the EBCDIC to ASCII translation along with some other essential logic. Because of this, we are able to simply FTP the VSAM backup files to the Unix machine in binary mode and invoke the same script we use for the human resource data. Again, the script preserves the grants, constraints, and indexes; truncates the tables when necessary; runs the loader; and then restores the grants and constraints and rebuilds the indexes. The FTP is controlled by the production job scheduler used in our MVS environment. We get the files after each day’s batch processing concludes. We use CRON to initiate the script on the Unix machine.

**Student**

Our student systems pose a greater challenge. The data exists in MVS IMS (hierarchical database) and includes all student and course information for more than twenty years. The native IMS files are not amenable to straightforward Oracle loading, and our efforts to use extract programs similar to the human resource system have bogged down for reasons related to manpower and machine power. We had to find a new strategy.
The method we have settled on uses the IMS reorg unload files. These files are produced monthly for performance optimization. They contain an ordered set of IMS segments for each IMS database in our student system. We take the dump files and FTP them to the Unix machine. Again, it is a binary transfer. We have written a "C" program that reads an unload file and the Mark IV data definition file associated with the IMS database. It loads all data associated with an entity (student, course, section, etc.) into memory and writes a set of records to a corresponding set of files. The files map to the tables in the warehouse and are ready to be loaded.

Our first attempt at this process simply used the IMS segments for the warehouse tables. While this would have been useful to people already familiar with the IMS data, we concluded that it would be worth doing some data modeling to simplify the design. The entire IMS student system has about 225 segments. A quick inspection made it clear that we could hold the data in half as many tables. Since we read all data for an entity into memory we can be very flexible in how we group fields into records. At this point we are designing a new relational layout. Our goal is to switch to a design provided by the vendor of the system we buy to replace our existing systems (PeopleSoft, SCT, TRG, etc.).

Our reliance on the IMS unload file is fine for periodic bulk loads of the entire data set. The IMS reorg process is run monthly and for our largest campus it is a 45 minute MVS job. We can run it daily, or weekly, but we won’t get the file on the Unix side until about 4 A.M.. With over 2 gigabytes of data for just one campus we do not have enough time to drop all the tables, reload the data and build indexes. For some purposes a weekly wholesale update will be acceptable, however, there are a number of users who have higher expectations. We have to find a way to identify just the data that has changed.

We are running IMS with only partial logging and we are unable to easily identify the updated segments on the MVS side. We are currently working on a strategy where we will write a daily checksum for each entity. Thereafter, each time we process the file we will recompute the checksum and compare it to the prior run. If they are the same we know the entity has not been updated. If the checksums differ we will replace all records corresponding to the entity. Because there is always a tiny possibility that the checksum will fail to indicate a change we will do complete reloads on a periodic basis to insure the warehouse contains accurate information.

**Fund Raising/Development**

Our fund raising/development system is contained in an old, unsupported version of IDMS (a network database) on MVS. A reorg of this system can take up to a week of wall-clock time. An extract of just the biographical data for our largest campus is a thirty-hour ordeal. Identifying changed records within the IDMS environment is impossible. Fortunately, we are converting this system to client/server. The vendor is BSR with a product called Advance C/S. The platform is Sybase 11 on Windows NT. The warehouse plays a crucial role in the conversion and will continue to be used afterwards.
Because we have made a large investment in Oracle we were reluctant to implement the BSR system in Sybase. However, the pressure to get out of IDMS was greater than our resistance to Sybase so we have proceeded. BSR has committed to do an Oracle version and our plan is to make that switch as soon as BSR delivers. We chose to do the conversion work in Oracle instead of Sybase because our long term plan is to standardize on Oracle and we wanted to enhance our Oracle skills.

In preparation for the conversion we created Oracle tables that match the BSR-Sybase tables and built a five step process for migrating the data. Step one is a set of MVS Cobol jobs that navigates the IDMS network an entity at a time. The programs write “lumpy” records of up to 32K (an MVS limitation) containing repeating elements of data pertaining to an entity. If an entity requires more than 32K we use a continuation indicator and write additional records. The resulting files (5 containing biographical data, 17 containing gift history data) are FTPed to the Unix machine.

Step two is a “C” program that “busts” these files into true “smooth” flat files. Step three loads these files into the Oracle tables that correspond to the IDMS data arrangement. Step four is handled by a collection of Oracle procedures written in PL/SQL -- one procedure for each table used by the Sybase application. The procedures select data from the raw IDMS arranged tables and insert records into Oracle tables that correspond to the BSR Advance C/S tables.

Step five simply copies the data from these Oracle tables to their Sybase counterparts. This can be done via tools that can connect to both Oracle and Sybase such as Oracle’s Gateway product or PowerSoft’s InfoMaker pipeline, or it can be handled using an unload/load mechanism. We are doing the latter with a Perl script executed on the Sybase machine. It does a remote procedure call to the Oracle machine to run an unload process, it then FTPs the result file to the Sybase machine and uses Sybase’s BCP loader to put the data into Sybase.

After the data is copied to Sybase we retain the copy in the warehouse for ad hoc reporting and other data warehouse activity such as Web access. Once the conversion of the application is complete, and we are done with IDMS, we will reverse the direction of the step five copy and continue to hold this data in the warehouse.

**Distributed Model**

At this time our HP 9000 holds data for all campuses. For performance reasons and to reduce our dependence on our wide-area network links, we place copies of some campus specific data on separate machines on three of our four campuses and on a machine at our hospital. Our fourth campus was in the warehouse business using Informix years ago and continues to go its own way. Four of the systems use Oracle on Unix - 2 HP 9000s, 1 IBM RS/6000 and one SGI Challenge. The other campus uses Oracle on Windows NT running on a dual processor Intel machine. Adding the NT environment required rewriting some shell scripts in Perl but has otherwise not been too burdensome.
Reproducing parts of the warehouse on each campus allows us to use less powerful or non-dedicated machines. It also relieves us of the burdens of account administration for a large population of users we do not know. The IT personnel on each campus are much better positioned to make judgments about what data any individual should be able to access. It makes sense to place account management responsibility in their hands.

**End User Access**

We have encouraged the use of PowerSoft’s InfoMaker reporting tool. We have conducted training for both the tool as well as data specific training and we have organized a user group that meets periodically to share experiences. InfoMaker reports can be distributed in a run-time form that will prompt for replaceable parameters. We are amassing a library of these reports that we are able to distribute to people who can’t, or won’t, take the time to learn to write their own. We chose InfoMaker over Oracle’s tools because it can go against Oracle, Sybase, and Informix (we have all three). It also had features not present in Oracle’s offerings when we made the decision.

**The Web**

In spite of the easy-to-use tools and all the effort we make to encourage users to take responsibility for meeting their own reporting needs, the number of regular users of our warehouse remains small. In an effort to deliver service to a wider audience we have built a number of Web interfaces to the data. The most sophisticated among these is a financial reporting system that will soon substitute for the long standing practice of monthly paper report distribution. We are now able to deliver nearly all of our standard accounting reports via the web.

These are not simply static renderings of the paper reports. Instead, we create on-the-fly, highly formatted, HTML pages that include a great many drill-down hyperlinks. The reports include account summaries, budgets, encumbrances, payroll, and transaction details. By storing month-closing snapshots for all accounts we are able to create these reports for any month going back to March of 1995. The Web interface makes all of this data valuable to a mass audience and is making microfiche obsolete.

Below are some representative screens from our financial reporting system. Figure 1 shows the starting screen which accepts an account code and optionally a month and a report type. The default is a summary report showing current month-to-date. Pressing the **Go!** button returns the report which is depicted in Figure 2. If the account is not known you can pick a campus and then select a division followed by a department. This will display a list of all accounts owned by the department. Picking an account at that point displays the current month-to-date account summary. A third means of access is a search mechanism that can directly retrieve data via such things as check numbers, invoices, vouchers and vendors - by name or number.
Figure 3 shows the attributes for an account. All of the reports use HTML frames which are problematic to print. There is a link at the bottom of every page which causes the same data to be displayed without frames. This format is printable. Figure 4 shows transaction details for the selected month. The column headings are buttons that cause the page to re-display with a new sort order. There is another report that will display all transactions on an account for the year-to-date.

The code behind the system consists of about twenty Oracle PL/SQL packages. Each package contains from 1-5 procedures. It took three people contributing about half their time for a couple months to produce most of the core system. We continue to enhance it by adding new reports. The rest of this section provides more detail on the technology we used.
Oracle-Web Connections

There are a number of competing technologies for connecting Web browsers to databases. They range from simple CGI scripts, to compiled, CGI-invoked, executables; to Java applets that make direct connections from the browser client to the database. Some of these are available at no cost, e.g., OraPerl and other Perl database extensions. Others, such as SapphireWeb, are commercial offerings.

In addition to technologies that do live database reporting to the Web, there are also products that perform mass, scheduled, HTML report generation. These reports are stored as ordinary Web pages and are then served up like any other Web document. IQ/LiveWeb is one example of such a product.

There are good and bad aspects to any of the choices. Because we have standardized on Oracle we felt comfortable using tools that only exist in the Oracle environment.

WOW, MOWI2, Oracle WebServer 2.0

In December of 1994 Oracle began distributing a free Web connection package called WOW. This package evolved into a product called Oracle WebServer that sells for $2500. The original distribution included the “C” source for a CGI style interface to the Oracle database. A programmer at the University of Missouri named Andrew McAllister took the Oracle freebie and re-wrote it in Pro*C (Oracle’s “C” precompiler) to fix a number of bugs and to add new features. He called this package MOWI (McAllister’s Oracle Web Interface). More recently he has re-written MOWI in straight C (distinct from Pro*C) using the Oracle OCI (Oracle call-level interface). He calls this MOWI2 and is selling it commercially. (http://www.mowi2.com)

Like WOW, WebServer 1.0 and Perl techniques, MOWI2 is invoked as a CGI script. However, MOWI2 is also able to run in a daemon mode. This mode eliminates the overhead of creating a large Unix process, connecting to the database, dropping the connection, and terminating the process. Once the daemon is started, the CGI script simply passes it the calling URL which it uses to launch the appropriate Oracle process. The returned data is fed straight to the Web Server (HTTPD daemon) which sends it on to the browser.

Because MOWI2 is launched via CGI, as opposed to being a native part of the Web Server, it still has some overhead for starting and stopping the CGI script as a Unix process. This design allows it to work with any WebServer that supports CGI (we use Apache). Other solutions, like Oracle’s WebServer 2.0 or Microsoft’s IIS, are able to eliminate all of the CGI overhead by becoming completely integrated into a Webserver.
**PL/SQL**

With both MOWI2 and Oracle’s offering the work (data retrieval and HTML markup) is done by stored procedures written in PL/SQL (Oracle’s internal procedural SQL). PL/SQL is well suited for data retrieval tasks and also supports enough flow control and data manipulation to handle just about anything we have tried to do. To make coding the HTML markup easier both systems include a collection of wrapper functions to handle the HTML tags. These functions are contained in Oracle packages called HTP and HTF. The set contained in HTP are Oracle procedures that return no value while those contained in HTF are functions that return a value. As implemented, most of the HTP procedures simply call their HTF counterpart which handles the logic and HTML markup. The value returned by the HTF function gets passed back to the browser.

If a programmer is already familiar with HTML coding it is easy to avoid using the HTP/HTF packages and write PL/SQL that outputs straight HTML. For someone new to HTML the mnemonics used within the HTP/HTF system make coding high quality Web pages very easy. We have programmers who have successfully written Oracle/Web interactions with no HTML experience.

PL/SQL procedures are compiled and stored within the Oracle database. They can be invoked by passing a URL to the Web server which includes the MOWI2 shell script (mowi) in a valid cgi-bin directory. In addition to the MOWI2 shell script the URL must include the Oracle account name in which the procedure exists, as well as the package name if the procedure is part of a package, and the collection of arguments expected by the procedure.

If the HTP/HTF packages are installed in an account called Web, and if the Oracle account used by the MOWI script has permissions to execute this package, the following URL would cause the words "Hello World" to be displayed in the browser.

http://www.host.domain/cgi-bin/mowi/web.htp.p?text=Hello+World

If the MOWI script uses the web account to connect to Oracle, or if there is a synonym for HTP visible to the MOWI account, the word web can be omitted from the URL. All of this is easier than this prose makes it sound. See the appendix for some examples of HTML markup in PL/SQL.

**The Legacy Database**

With the warehouse in place and with good tools for handling web-database interactions, we have gone the next step of building applications that capture data in web forms. We store the data in the warehouse, and we return daily batch updates to the legacy databases.
Change of Address

A simple example of this kind of application is the change-of-address form used within our human resource system. The warehouse contains all of the addresses for any employee. It is a simple PL/SQL procedure that takes an employee ID as input, and displays the set of addresses for the employee in a Web form. After the employee makes the desired changes and hits Submit another PL/SQL routine performs some simple validations of the data and if it all checks out it updates the warehouse copy of the employee’s addresses.

Any record which has been updated is marked for migration. Each evening CRON executes a Pro*C program that reads the flagged records and creates a file which is FTPed to the MVS machine. The Pro*C program alters the flag to indicate the change has been migrated. An MVS batch job executes and updates the records in the human resource system. All activity is time stamped to permit error recovery.

To handle security we allow each employee to self-register for a PIN which together with their employee ID number serves as our authentication mechanism.

Other examples of the use of this mechanism include forms for the W4, vacation sick leave, money received reports, and journal entries - all built or in progress. Our ambitions extend to requisitions, travel reimbursement, encumbrances, and even vouchers and expense reimbursements.

Cash Receipts Report

Our most widely used Oracle/Web system is our cash receipts report. Across the University of Missouri there are over 600 locations that take-in money. They range from large volume operations like the bookstores and hospital to the once-a-year subscription sales for an academic journal. The traditional means of recording income was a paper form that was delivered to an accounting office for data entry. This procedure had days or weeks of built-in delay between the time the money was accepted, deposited in the bank, and finally reflected in the accounting system. The bank-to-book reconciliation was very labor intensive and could be done monthly at best.

The electronic cash receipts report system allows a department to collect income, bank it, and record it on the same day. It avoids the redundant data entry inherent in paper forms. It also makes the bank reconciliation much simpler and allows most of the reconciliation to be performed daily.

The system accommodates locations that do their own bank deposit, those that take their cash receipts to a cashier office, as well as electronic fund transfers. There is a web-based security tool that allows the accountants to control access to the various components. This includes limiting who can perform a direct deposit. This is important because with a direct deposit there is no opportunity for any kind of approval. A Web based direct deposit form will update the live transactional system the day it is entered. On the other hand, anyone in the University is allowed to prepare a cashier deposit. These forms wait in a queue pending approval from a cashier. This open-door policy
minimizes the administrative overhead. The accountants are only concerned with managing the employees allowed to do direct deposits.

The following figures depict two of the Web pages that drive the system. The basic data entry form is used by all preparers. There are sub-systems for the cashier’s office where deposits are approved as well as a maintenance tool that allows the accounting offices to track activity.

![Figure 5](image1.png)  ![Figure 6](image2.png)

The data entry form uses javascript for on-screen arithmetic and some data validation but all of this is checked again at the server before data is saved. The evaluation at the server also includes checks for valid account codes and subcodes. This is an improvement over the paper based process where the data entry clerks had to accept what was given to them, send it back, or make a phone call before they could correct some items.

**Conclusion**

The decision to implement some of these systems is closely coupled with the timetable for replacing our administrative systems. Most of these Web form applications can be knocked off fairly quickly. A useful life of 2 or three years can justify the effort. The real limiting factor is having people with skills and time to do the work.

The University of Missouri information technology staff has a wealth of experience. The vast majority of this skill centers on the technologies and systems we have used to support our business for the past twenty years. We have people with skills in MVS, JCL, Cobol, Mark IV, and IMS. However, these same people are weak in RDBMS, SQL, Unix, distributed processing, and clustering. Most people have been so busy doing their jobs that there was little time left over for learning new skills that didn’t have an immediate return in their day-to-day activities.
The unbelievable changes that have occurred in the past decade in microprocessors, operating systems, and relational databases have had a tremendous impact on the academic curriculum, but these same transformations have hardly touched the way we keep the books, do the payroll and print the transcripts. A freshly minted computer science graduate has the technical skills that are in demand but they too often lack the life experience to understand the issues and are too often unable to see the scope and complexity of the systems we are responsible for maintaining.

Part of our motivation for warehouse and web projects is to expose the IT staff to the new technology. This is a tough item to work into any cost justification formula but it is a legitimate and necessary investment. As we plan for the replacement of our legacy systems we must also plan for the skill training to make the conversions successful. The projects we are working on are designed to deliver benefits to the users but we are also benefiting from the opportunities they provide for our Cobol programmers to learn SQL.
Appendix

Below are examples of Oracle PL/SQL stored procedures that make use of the HTML wrapper functions. They would be called from a web browser with a URL of the following form:

http://www.server.domain/cgi-bin/mowi/hello_world

or with an argument as in the last example:

http://www.server.domain/cgi-bin/mowi/data_retrieval_loop?dept_key_input=U14008

The wrapper functions accomplish the HTML markup by concatenating HTML code and tags to text strings. The tags are specified as arguments to the wrapper functions.

```sql
--Procedure demonstrates the basic text output mechanism.
CREATE OR REPLACE PROCEDURE Hello_World IS
BEGIN
  htp.p('Hello World');
END;
```

```sql
--This demonstrates the use of string literal
-- HTML codes or the HTP/HTF wrapper functions
CREATE OR REPLACE PROCEDURE Hello_World_bold_twice IS
BEGIN
  htp.p('<B>Hello World</B>');
  htp.nl;
  htp.bold('Hello World');
END;
```

Figure 7

Figure 8
-- A more complex example that displays as a full-width table
-- with two rows and one cell per row. The two rows use
-- different procedures for handling the HTML cell codes.
-- The first method can more easily handle complex
-- cell contents. The second method uses fewer statements.

CREATE OR REPLACE PROCEDURE
Hello_World_in_a_table_twice IS

BEGIN
  htp.bodyopen;
  htp.tableOpen('WIDTH=100% BORDER=2');
  htp.trowopen;
  htp.tcellopen('ALIGN=CENTER');
  htp.p('Hello World');
  htp.tcellclose;
  htp.trowclose;
  htp.trowopen;
  htp.tcell('ALIGN=CENTER', htf.bold('Hello World'));
  htp.trowclose;
  htp.tableclose;
  htp.bodyclose;
END;

-- This example shows how to retrieve data using a SQL cursor and output it in a table with
-- invisible border lines. The table has as many rows as the cursor returns. Each row has two
-- columns. The first column contains the name returned in the cursor. The second column
-- displays the on campus address. The procedure requires an input argument which is also passed to
-- the cursor. The cursor performs a join and returns all records for the input department.

CREATE OR REPLACE PROCEDURE data_retrieval_loop (dept_key_input IN VARCHAR2) IS

CURSOR get_data (dept_key IN VARCHAR2) IS
  SELECT a.name, b.address FROM hrs.emp_personal a, hrs.emp_address b
  WHERE a.federal_id_number = b.federal_id_number AND a.department_key = dept_key
  AND b.address_type = 'ON_CAMPUS' ORDER BY a.name;

BEGIN
  htp.bodyopen;
  htp.bold('These are the people in department ' || dept_key_input);
  htp.nl;
  htp.tableopen('WIDTH=100%');
  FOR I IN get_data(dept_key_input) LOOP
    htp.trowopen;
    htp.tcell('ALIGN=LEFT', I.name);
    htp.tcell('ALIGN=LEFT', I.address);
    htp.trowclose;
  END LOOP;
  htp.tableclose;
  htp.bodyclose;
END;