

**EXPERIENCE GAINED IN APT DATABASE DESIGN THROUGH THE  
DEVELOPMENT OF THE CALTRANS/UCPRC HVS DATABASE**

Jeremy D Lea  
University of California Pavement Research Center  
University of California, Davis  
Department of Civil Engineering  
One Shields Ave  
Davis, CA 95616  
Tel: +1 (530) 752-1752  
jdlea@ucdavis.edu

Submission Date: 29 February 2008

Words: 5410

Figures: 1

Tables: 0

Total Word Count: 5660

## **ABSTRACT**

Over the past twelve years Caltrans and the UC Pavement Research Center have operated two Heavy Vehicle Simulators. They have tested over ninety one sections, accumulating nearly 70 million passes, resulting in a 12 GB database with more data being added daily. This paper details some of the lessons learned in the design of the database and in the handling and workflow of data within the project. In that time three different database systems, four different data acquisition systems and several different processing techniques have been used. The current system is described, including data collection procedures, data processing, and the web based front-end, along with the reasons for the choices of software and database systems. The data has all been stored in text files, so that it can be manually edited and the entire database recovered in the event of database failure, and the format of these files has evolved over time. Best practice for the design of these output files is described, along with data table indexing and relationships for building a relational database around APT data. It has proved very worthwhile in the post-processing of the data to have strict rules in the database to ensure that the data is accurate and consistent. In particular, the data has all been indexed by both a pavement location and time/repetition index, using relationships which can be difficult to implement in some database software, but ensure that in the future the data will remain useful. Research, by its very nature, involves changing requirements and data needs, and this paper looks at how to balance those needs with the requirements for long term data stability to support large scale efforts, such as the calibration of ME design methods.

## **INTRODUCTION**

In 1994 Caltrans purchased two Heavy Vehicle Simulators (HVSs) from the CSIR in South Africa. Since 1995 these two machines have been operating almost continuously in California, and tested over ninety one sections, accumulating nearly 70 million axle passes, resulting in a 12 GB database, with more data being added daily. In that time three different database systems, four different data acquisition systems and several different processing techniques have been used. This paper details some of the lessons learned in the design of the database and in the handling and workflow of data within the project. The most important of these lessons is that the process of APT testing must be seen as going from test section to database to reports, not test section to reports to database. Research, by its very nature, involves changing requirements and data needs, and this paper also looks at how to balance those needs with the requirements for long term data stability to support large scale efforts, such as the calibration of ME design methods.

The current project workflow and database system is described first, including data collection procedures, data processing, and the web based front-end, along with the reasons for the choices of software and database systems. Over the course of implementing the new database, and writing the loader to handle all of the existing data and the variety of data expected going forward, a number of problems had to be addressed and overcome. The goal of this paper is to lay out some of the lessons learned in this process, in the hope that they are of some assistance to others developing APT databases. Accordingly, this paper does not go into detail about the exact table structure of the database, since one of the lessons is that no two projects' data is alike, and developing a 'universal' database for APT will only mean that it will not work for anyone. More details on the database can be found in the report by Lea and Popescu (1).

## **BACKGROUND**

The Cal/APT project began in 1995, with the goal of providing pavement research support to the California Department of Transportation. Since then the two HVSs, now run under the auspices of the UC Pavement Research Center, have tested 91 sections across seven different sites around the state, with a wide variety of different materials. In the process over 350,000 readings have been taken.

The initial data acquisition systems were supplied by the CSIR in South Africa and were based on the systems used successfully for over the 25 years in South Africa. These systems were based on custom hardware and software, and used MS-DOS – because that was what was available at the time. They produced output in text files, which were archived onto floppy disks and copied onto the project engineer's computer, where they were manually analyzed using spreadsheet software to produce the various charts and then analysis for the final report on the project. As part of the initial project a new database system was also developed by the CSIR, using the newly released Microsoft Access and Visual Basic. This database was used for a short while to store data from the project, but updating of this database often lagged behind testing, and, more importantly, the database was too large to move on a floppy disk, so had to be written to CD-ROM, and then installed onto the engineer's computer. As a result, it was not really used. This database was also implemented in the Oracle RDMS, running on a Sun server, but this was never deployed in California.

In 1997 the project began testing on rigid pavements, utilizing a number of instruments not normally used in South Africa, including strain gauges and the newly developed Joint Displacement Measuring Device (JDMD). The software and hardware was modified to use these instruments. However, the database was not modified, and it soon became obvious that the existing data acquisition hardware and software were not suited to the different needs of testing on rigid pavements, which required automated testing to capture hourly effects such as curling and warping, along with simultaneous collection of data from a number of different instruments.

As a result, a new data acquisition system, based on Labview<sup>®</sup> was designed, and implemented from 1999. There were some teething problems with this system, as is common on most systems developed from scratch, and the software only stabilized by 2002, when all of the old data acquisition systems were replaced by the new system. The system has been fairly stable since 2002, with only minor electronics changes to improve filtering.

This new system uses a Comma Separated Value (CSV) output format, which was easier to read using Excel, but harder to read by humans, since the data is not aligned, and also contains some redundant information. While the old software had read 256 points for each deflection bowl or profile, the new software reads a variable number of points, with a lower spacing, and so records considerably more data.

More importantly, these files could not be read by the old database software, and the database structures, which were designed to hold the information from the old CSIR style text files, could not contain all of the information from the new files. As a result, the database system needed to be fully redeveloped. This process began in 1999, and was also completed in 2002, and is documented in the report by Lea and Popescu (1), which was published in 2003. Since then the database components have been slightly updated to handle some edge cases and the web based interface to the database has been significantly improved.

## **DESCRIPTION OF THE CURRENT SYSTEM**

The current data acquisition system uses a combination of National Instruments and custom hardware, with a Labview<sup>®</sup> based front end. This system allows for the collection of all of the various electronic instruments used directly in HVS testing by the UCPRC, including the Multi-Depth Deflectometer (MDD), Road Surface Deflectometer (RSD), Joint Displacement Measuring Device (JDMD), Crack Activity Meter (CAM), strain gauges, pressure cells and the laser profileometer. Besides the profileometer, whose operation is not dependant on the HVS to provide a load, all of the instruments can either be read at regular intervals during normal trafficking of the section, or manually with the wheel moving at creep speed (which is the traditional way of doing HVS measurements). The instruments can also be read while the HVS wheel is not moving to record environmental influences. The system also records temperatures simultaneously from a range of thermocouples, either at the same time as the measurements are taken or at hourly intervals.

The software allows the operator to inspect all of the collected data for each reading, and either accept or reject the reading (in case of some error, like incorrect instrument placement). Once the data is accepted it is written to disk, in data files which are named by the section being tested, the type of instrument (since each instrument has a slightly different data format), and the current trafficking repetition. At the end of the data collection, or each morning if the data collection is automatic, the operator copies the data files from the DAQ computer and emails them to the project engineer and the database maintainer.

The project engineer takes the data files and loads them into the database using the 'loader', which is an Excel spreadsheet with some fairly complicated macros, which checks the files and processes them. The engineer can then examine the data, and decide if it is acceptable (requesting that the data collection be repeated if it is not), and then produces some summary information which is emailed to the various parties who are watching and waiting for new data, normally with a brief comment on the data and the progress of the test. This data is also immediately available to all via the web based front end. As the data is loaded, the loader also moves the successfully loaded files to a special directory where they are archived and backed up nightly.

At the end of the test, the data is extracted from the database into spreadsheets where it is then plotted and manipulated to produce the final charts and other information needed for the first level analysis reports on each test section. There is also a program which can extract data from the database into a format usable by the CalME pavement design software, where the measured data can be compared to the calculated performance, and calibration of the various ME design models performed.

The loader forms the heart of the database system, and it operates by first looking over the files it has been instructed to load, and only loads those not already present in the database (since the file names are constructed to be unique, it can tell if a file has already been loaded). For each file it then opens the file and determines the instrument type and data format (the loader can load both the old CSIR and new CSV formats), and then proceeds to loop over each block of raw data in the file, first computing the header information (the metadata – data about the data), such as the time of the readings, repetition, load, wheel configuration, etc. Some of this information is not dependent on the instrument and so is loaded into the general section information tables in the database.

The raw data is then loaded, and the data processed through a complex filter, described in (1), which performs a number of checks on the raw data, and might perform smoothing of the data. Each piece of raw data is then loaded into the database, and the loader proceeds to the next block of data.

If an error occurs then all of the data from the file is removed from the database and a message is recorded detailing the error, so that the engineer or the database operator can fix the error. The most common errors are those caused by the operators, such as restarting readings, so that the wrong repeat count is recorded (e.g. blocks numbered 1,1,2 instead of 1,2,3), or header data which does not match with that previously recorded (like incorrect trafficking repetition).

The loader is normally used to load data into a PostgreSQL database, which is used by the web based front end, or a local Access database, which is useful for testing and also for times when work needs to be performed offline. An Oracle database was used initially, but the license fees required to run an Oracle database with a web based front end were exorbitant, and so the open-source PostgreSQL database was selected, since it has all of the useful features of Oracle, but requires considerably less memory and operator training, besides being free. For this kind of system it is actually faster than Oracle, and almost as fast as Access.

The web based front end is written in PHP, and outputs the data in HTML and charts using Scalable Vector Graphics (SVG), which is a vector graphics drawing format for the web which has recently been gaining support, although it lags behind Adobe's Flash format. It has the advantage that it can be output from a scripting language like PHP, where Flash requires custom tools. SVG is supported natively in Firefox, Opera and Safari, but is not supported by Internet Explorer, so a plugin is required to view the files.

## LESSONS LEARNED

### The database is the direct output of APT

The first major lesson was that the database of results cannot be an afterthought in an APT program. Since the project engineers were originally working with the raw data in spreadsheets and using that to produce analysis reports, the database was seen as a ‘nice to have’ item, in case anyone ever wanted to go back and look at the data. The reports on the various sections were seen as the project deliverables, and it was perceived that these contained all of the information needed for any future re-evaluation, or for building higher level conclusions.

The issue here is that, while the first level analysis reports are critical to understanding the results from any particular test, and **one should never try to use APT data without first reading these reports**, going back to these paper (or electronic paper) reports to determine all of the information for a section means that this cannot be done on an automated basis for activities like calibration of ME design procedures. Even if all of the data is in spreadsheets, differences between various engineers mean that it is difficult to (a) find the latest data and (b) format the data in a uniform way for a comprehensive analysis. However, it is this process – of using APT to calibrate models – where the real payoff for APT lies. APT projects are traditionally motivated and funded using short term goals, such as ‘will asphalt A perform better than asphalt B?’ The experimental design is set up to answer this question, and the reports (hopefully) provide some answer. In this process, the collection of data along the way is sometimes superfluous, since the engineer can often come to the same conclusions by just looking at the pavement. It is often hard to claim the benefit from drawing these conclusions, by using APT at this level, as a tangible benefit from APT when doing cost/benefit analysis of APT testing, because more often than not, the next phase is to move to field trials, and the outcome of these is the real litmus test for performance. Many critics have argued that all APT testing does here is delay the field trials and final adoption of the product.

However, it is at the next stage, after a successful field trial, that APT testing has the most value. A field trial normally takes place under semi-controlled conditions, and seldom provides enough information to validate a design procedure for the new material. To do this, a combination of APT and lab testing data is needed, and it is this process, of getting the new material into the design manuals, which actually allows it to be used in standard practice. Without usable data at this stage, the product will remain trapped in the ‘promising but needs more investigation’ category.

While the first level report might be what is necessary to get the project signed off, so the funding is paid, the database must be seen as an integral part of the analysis of APT tests. To do this it is necessary that the operators, the project engineers and the client, see the process of APT testing as going from test section to database to reports, not test section to reports to database. If this is ingrained into the thinking, then, for example, the operator will stop the test if the instruments are not working, rather than saying ‘well we’re running, so we’re damaging, so we’re doing our job’, which then becomes a statement like ‘the gauge wasn’t working, so we did our best to recover some data, and so our conclusions are preliminary and we need to do more work’ at a conference like this. This also helps encourage the project engineer to work from the database in the analysis, rather than using raw data. The result of this is that the database is always kept up to date, and the project engineer can ensure that the data paints the same picture of performance that they are seeing ‘on the ground’.

One simple, although heavy handed, way to accomplish this is to highlight the cost of taking readings, rather than the cost per load pass. Using a coarse calculation the Caltrans HVS projects cost about \$0.25 per loaded pass. But the cost of a single reading from an instrument is about \$50 (making the traditional 'full set of readings' on an HVS test worth about \$15000). This causes the operator and the engineer to be data focused rather than loading focused. Rather than thinking 'we're not running, so we're losing money', it is better to think 'we're not recording good data, so we're wasting money'. Obviously, if the machine is not running then you are neither trafficking or collecting data, so this still means that machine reliability is an important consideration since there are always fixed costs associated with an APT operation.

### **Always use the database**

During day to day operations with an APT test it is tempting to use the raw data that is collected, and pull it into spreadsheets, which can be easily updated and changed, and to make quick graphs and tables to track the visual performance of the test. Generally, these spreadsheets morph into a final data analysis for the project, and are then copied and recycled for the next project. From the project engineer's perspective, this provides a simple day-to-day workflow, with no major downside. However, in this scenario the database will be left to lag because no one is using it. Worse still, any incorrect information from the raw data files (and this happens on the best of projects), will normally not be corrected, and the database will eventually get out of sync with the data used to perform the analysis, to the extent that the database becomes unusable. This trend has been clearly observed in our testing, where some project engineers have used this approach, and as a result the data for these projects has required significantly more post-processing to make it usable.

An additional problem is that the data processing in this scenario is not always consistent. As the spreadsheet migrates from section to section improvements are made, but the old data is not reprocessed, so the results for each section are subtly different.

This problem can be easily solved by requiring all work to go through the database. By having a stage where the data is loaded, checked and cleaned as soon as it is collected the engineers can catch errors before the data is used or reported. And, since the data is loaded using a uniform procedure, the analysis is always consistent. The engineers can then extract data from the database to work on, which requires a small amount of extra work over just opening a file in Excel, but this work can be programmed into macros and software, so does not slow the workflow. This also means some additional training for project engineers, to enable them to work with the database, but these are not difficult skills, and the advantage is that the engineers are able to perform smarter analyses on the data once they are able to interact with the database and perform queries to find the information they want. For managers and clients, a web based front end provides a quick way of keeping abreast with current progress, and does not require any additional training or software.

### **Use an intermediate format that is editable**

While it is tempting to conceive of a database that is directly linked to the DAQ system, in practice this has a number of drawbacks. The first is that the DAQ system needs to be physically linked to the database, which is presumably housed at the research offices. This might not be a problem for fixed facilities such as test tracks, where the data is being collected beside the

pavement, this is a problem for mobile facilities like the HVS and ALF. But even for a fixed facility, this means that network problems can prevent data collection and thus testing.

The main problem, however, is that databases are hard to edit, and back up incrementally. With an intermediate file, which is in a human-readable and editable format, errors in the data can be quickly corrected. In addition, text files (which includes formats like CSV), can be managed using a revision control system such as CVS or Subversion. These systems can keep track of these corrections, along with a log message about why the correction was made.

The CSV file format in use by the UCPRC is not ideal, although it is meeting current needs. If another major refresh of the system is performed, the files would be changed to include a magic starting line (like 'UCPRC HVS Data MDD v1.0'), which would indicate the kind of data (the system currently has to rely on the file name to find this), and the revision of the format so that the loader can chose the right code path. Following this, a comment line would be included which would allow for details from the revision control system to be included, which the loader could parse and store in the database, so that it could tell not only if it had loaded the file, but what version of the file it had loaded.

Following this a header block would be written, which would have any information which was guaranteed to be invariant between the various data blocks in the file, including section number and instrument locations. The data blocks would then follow, each with their own headers, such as the time the readings were taken, which instruments were recorded, etc.

A comma separated value (CSV) file is not ideal for this task, because major spreadsheets register this file format as one that they can handle, so users can click on the file to open it directly. However, no major spreadsheet preserves the formatting of the file, so if it is saved the entire contents of the file is rewritten, making it appear to a revision control system as if the entire file had been edited, even if no actual changes were made. A fixed width text file format is just as easy to read and process, but will naturally open in a text editor, which will normally not change the structure of the file upon saving.

One field which should be included in the header of every data block is a 'status' field which indicates whether the data in the block is good, questionable or bad. Often there are problems with instruments which are only noticed after some time, and it is wise to be able to keep this data in the database (to avoid it being repeatedly seen as new data), but still be able to hide it from normal queries or display it as an outlier.

### **Use strong relationships and enforce every assumption**

It is tempting in research databases to make the fields as general as possible, and to use the tables as if they were spreadsheets. In this scheme the relationships between various pieces of data is implied. It is assumed that because the data is loaded from data files, where it can only be one way, that these invariants are true. In the original HVS database, for example, the instrument layout for the sections was in a separate table from the data, so it was possible to add data for instruments which did not have any position information. While this should not be possible in practice, this happened, and it is difficult, after the fact, to figure out what went wrong.

In the current database, every relationship is enforced, and all the possible unique combinations of fields are defined as unique keys, so that it is impossible to add conflicting data. For example, each reading for each instrument type is provided with a unique auto-generated sequence number. But there are also two other keys: the combination of the section, instrument and the time the reading started (since it is not possible to read two different sets of data off an instrument at the same time); and the combination of section code, instrument, time when the

overall data collection began, the testing load, pressure and lateral position and the repeat (since it is not possible to record two first sets of data with the wheel at 40 kN and 720 kPa over the instrument). These additional keys are not needed by the database, and slow it down fractionally. However, it does catch errors. These errors would go unnoticed otherwise, resulting in data which was otherwise difficult to explain. These rules are adjusted for each instrument, to be as strict as possible. For example, the HVS software can only handle one profileometer, so the key here limits the readings to section code and reading time, since it is not possible to read two positions at the same time.

The HVS database makes use of time values as keys, in preference to repetition counts, because we can take two sets of readings at the same repetition (for example, before and after planned maintenance on the machine), and we can also record environmental data where a repetition count is meaningless. These fields are also stored efficiently by the database and are suitable for use in foreign keys defining relations between tables.

In addition, any fields which can only take certain values are constrained to ensure that only those values exist. This helps to ensure that the data remains consistent, although it does limit future values. An example of this is the wheel configuration. On the HVS we can test with dual or single wheels, with a variety of tires. These are hard coded into the database, so if someone decides to test with a new type of tire, then the database must be updated to allow this. This is actually a very simple process, and so it doesn't slow the workflow – but it does make the engineer stop to think about whether this is actually the right thing to do.

Some constraints are, however, difficult to enforce in a database, particularly in Access which does not support triggers. For these constraints the loader has a number of functions to test the data integrity of these rules, and to correct the database automatically if possible. Examples are these are the fact that the trafficking repetition count cannot go backwards with time, or that a set of readings must actually contain at least one reading. It is possible to flag suspect information this way, so that the engineer can quickly resolve the problems.

Having all of the data flow through the database also helps ensure data quality, because it means that most data errors are corrected before the engineer begins working with the data. In addition, the engineers come to expect the data to be good, and when they see problems while working on the first level analysis reports they are more likely to try to fix the data in the database rather than seek some quick workaround. Finally, having a quick turn around from data collection to usable outputs also means that problems tend to be corrected quickly, and often without loss of critical performance information.

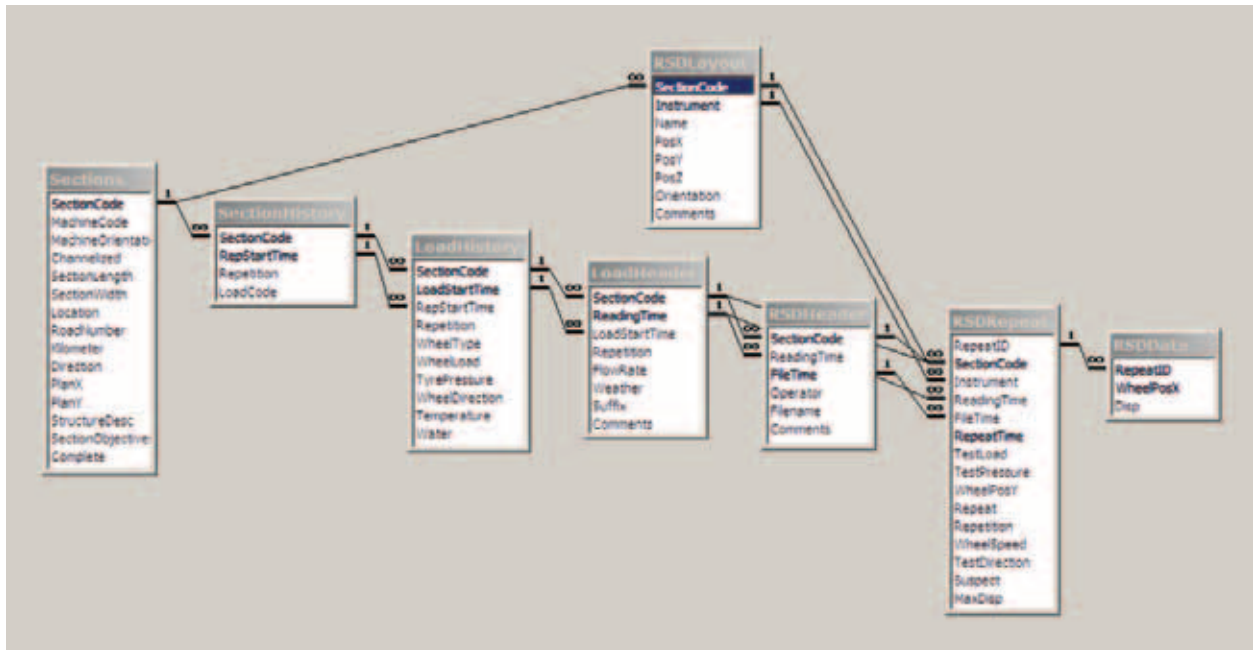
### **Use combined location and time based keys**

Building on the idea of using the strictest possible keys, the HVS database makes use of a dual hierarchy of relationships: one based the location of the instrument within the section and the other based on the time readings were taken. This violates a lot of what has been written about relational database design, and causes some problems for database software. Particularly, some versions of Oracle will not allow cascading updates on these types of keys, so these have to be manually implemented using triggers. Access can perform cascading updates, but not cascading deletes. This is seldom a problem, since this only occurs when an entire section is deleted, which is unusual.

These keys are critical to the correct functioning of the database, since they ensure that the instruments locations are not 'moving around' during the test (which can be done, but requires that the new location be given a new name, so that the data is not confused). They also

ensure that position information is always available for an instrument, without recording it in each record.

An example of this is shown in Figure 1, where the tables for the RSD are shown. The first four tables from the left are independent of the instrument, and store the section information, the traffic loading information (split across two tables to deal with some anomalies in the historical data), then a table of information about each block of readings (so that readings taken at different times, but logically simultaneous can be grouped). The top table, RSDLayout, holds the location information, the RSDHeader table holds the header from each file, and the RSDRepeat table holds the information for each actual reading. Finally, the RSDData table holds the raw data. The dual hierarchy of keys can be clearly seen from this figure.



**FIGURE 1: Example of database structure for the RSD**

### **Do not store raw data in individual records**

When the original database was designed, the DAQ system always output 256 points of data for each instrument, so the database had tables with 256 data fields to store this information. With the new system, there were a variable number of data points per bowl, and so it was decided to move to storing each data point as an (X, Y) pair. This works, but results in massive data tables for the raw data, which slows some operations considerably. It is easy to extract the raw data from these tables, but, when all is said and done the raw data is seldom needed in any context other than plotting the deflection bowls or profiles to show what they look like. All of the important summary information has already been extracted and stored. As a result, the queries almost always extract the full bowl at the same time. As a result, a better procedure is to store the raw data as text, using a CSV format, with one string of X values and one of Y values. This can be converted into data cells quickly using string manipulation functions, and plotted, without burdening the database with storing such large tables. Some databases also support string compression, so this further reduces the data size.

## Software documentation and training

It is customary for software products to require a written manual, and some documentation about the design of the product. In our experience with these various systems, the written manuals have never been consulted, and they are generally inaccurate or not detailed enough. Basically, they tend to be pages of screen captures which cover the obvious, which engineers accustomed to computers do intuitively. There is thus little point in producing such documentation.

However, the languages chosen for the current system, Labview<sup>®</sup> and Excel/VBA are not compiled and so the source code is open for engineers to look at, and modify if need be. These have been documented with internal code comments, which attempt to describe the operation of the system – although the actual code really is the final authority on what is happening. It is not recommended that a ‘black box’ approach be used for this type of work – the engineers involved will inevitably want to investigate the operation of the code and offer suggestions or tweaks, especially as new instruments are being developed. If the software was purchased as a black box from a vendor, then these will not be possible.

Finally, the key to the success of implementing such systems is training. If the machine operators, engineers and other staff involved are not aware of how the system works, in detail, then they will not be able to do their jobs properly. It is thus vital that new staff are given careful, on the job training, and that they feel comfortable calling those more experienced with the system when they have questions, rather than blindly forging ahead.

## CONCLUSIONS

This paper has detailed some of the lessons learned in the design of a large APT database and in the handling and workflow of data within the project. While not all of these lessons are applicable to all facilities, it is hoped that these will help those just starting out on the road of using APT data in the calibration of pavement design methods to avoid some of the pitfalls.

The most important consideration is that the database must form an integral part of the workflow in the day-to-day running of an APT project, and not be an add-on deliverable. This is especially true if the data is going to have long term value. To this end, it is important the database not be ‘worked around’, but rather adapted to the changing needs of the project. Because this means that the database becomes a working entity within the project, it argues against standardized databases across various APT projects. By making the database an integral part of the workflow, the quality control of the data also becomes a standard part of the day to day operations, rather than the guessing game required if the data is only processed years later.

In terms of the design of the system it is, however, important to have the primary storage of data be in human readable and editable text files, which are used to populate a relational database, because this allows a clean separation of the data from the form of the database, and protects against database corruption and general difficulties with managing a large amount of information in a relational database. This intermediate representation is, in any case, necessary if the DAQ system is not loading data directly into the database. The database should then have as strong a structure as possible, to ensure that the information being loaded into the database from these files can be verified and any errors corrected.

## ACKNOWLEDGMENTS

The work described in this paper was performed as part of the APT program being conducted by the University of California Pavement Research Center (UCPRC), with funding by the California Department of Transportation (Caltrans), Department of Research and Innovation. However, the views expressed are those of the author alone.

## REFERENCES

1. Lea, J.D. and Popescu, L. 2003. *The Design and Implementation of the Pavement Research Center Heavy Vehicle Simulator Database*. Draft report prepared for the California Department of Transportation. Pavement Research Center, Institute of Transportation Studies, University of California. UCPRC-RR-2003-01.