Paper 018-2013

Extraction, Transformation, and Loading (ETL) for Outcomes Measures of Workers' Compensation Benefits

Mike Maier, Oregon Department of Consumer and Business Services

ABSTRACT

Base SAS® was used to create a data sub-system for measuring outcomes, added to a data system (coded in SAS) of benefit costs and employment. One claim per injured worker per fiscal year is extracted as a study or control record, using business-rule code. Disability benefits and employment data are transformed to time-series records for claims, which are transformed to time-series statistics by fiscal year. Programs are run remotely on a UNIX data warehouse, and SAS data sets and metadata are loaded to the warehouse and downloaded to a LAN. Quarterly generations are kept for analysis of claim development.

INTRODUCTION

The Department of Consumer and Business Services (DCBS) is Oregon's largest business regulatory and consumer protection agency, and it is the regulator for workers' compensation insurance and the benefits paid to injured workers from that insurance. Among the services that the Information Technology and Research Section provides for DCBS is development and maintenance of data systems for workers' compensation claims and reports about costs, benefits, and performance or outcome measurements. Since the 1980s, analysts have relied more and more on Base SAS® and sometimes SAS/STAT® code for extraction and transformation of operational data to load statistical reports, data sets, and metadata. For the last decade or so, programs are stored on a LAN and run on a UNIX server and warehouse via SAS/Connect®. The primary source for data has become Oracle databases made available through SAS/Access®, and SAS/Share® is also part of the architecture for serving data to multiple SAS users.

Indeed, Oregon policy makers have shown continuing commitment to data-driven decisions and interest in measures of workers' compensation system performance. One such measure, from the Oregon Workers' Compensation Premium Rate Ranking Study, has become a national resource for interstate comparison of insurance rates paid by employers. Two performance measures for Oregon's return-to-work programs have been reported to the Oregon legislature for over a decade. Recent work has refined those measures of wage replacement and employment retention toward claim-outcome statistics, which serve as companion and counterpoint to the rate-ranking statistics.

Both the return to work and the claim outcomes measures represent adaptations of techniques used by the Rand Corporation, Upjohn Institute, and others to compare wages and disability benefits of injured workers to the wages for a control group. Both sets of measures are made possible by a SAS application that creates a data system for benefits and wages paid to injured workers. Application development began more than a decade ago with the nearly concurrent provision of flat-file data written to specifications by external researchers: RAND, for derivation of wage-replacement statistics, and Michigan State University, for analysis of changes in the cost of workers' compensation insurance in Oregon. Today, that application is a highly integrated set of programs, data sets, and statistical reports.

Fiscal year	Sequential relative-injury quarter	Wages, percent base wages	Disability benefits, percent base wages
2001	0	76.8	21.4
	1	75.4	15.4
	2	74.8	12.0

Table 1. Sample Claim Outcomes Statistics

Table 1 shows some statistics that may be generated from the claim outcomes data, covering the injury quarter (quarter 0) and the two subsequent quarters, for claims with injuries during fiscal year 2001. In the quarter of injury, workers earned almost 77 percent of their base or pre-injury wages, and the disability indemnity benefits paid came to about 21 percent of their pre-injury wages. The focus of this paper is the extraction, transformation, and load of the claim outcome data—the SAS code and programs that make these statistics possible.

THE WCBEN SYSTEM, BENEFITS AND WAGES PAID TO INJURED WORKERS

"Data preparation is the big thing…" for successful performance measures and sophisticated analytics (Siegel; used by permission). The operational and historical data about Oregon workers' compensation claims and disability benefit payments reside in many data tables in three Oracle data bases, flat files of legacy data, and spreadsheet responses to data calls. Employment and wage data are provided to DCBS as flat files in federal government-defined format. Not surprisingly, variables with similar meanings have differing attributes: dates may be date-time, numeric, or character, for example. Most variables of interest have minimal data reporting and entry edits; data quality is often

marginal, especially as analysis extends down to finer levels. From those disparate sources, a couple dozen SAS programs, two or three thousand lines in total, provide the ETL to the WCBEN data system of SAS data sets and standard reports. These programs are organized into sub-systems, groups of programs with a common theme and purpose:

- CSC signifies a claim and its status;
- CST is zero-to-many (usually many) transaction records of benefits, employment, and wages for a claim;
- CSY summarizes many claims; and
- CSO is the new outcomes measures.

Figure 1 is a simplified and partial flow chart of the extraction and load for the first three data sub-systems. Not only is data preparation a big thing, it's also not very pretty. This diagram is meant to convey the many dependencies, with one implication being lots of room for programming errors.

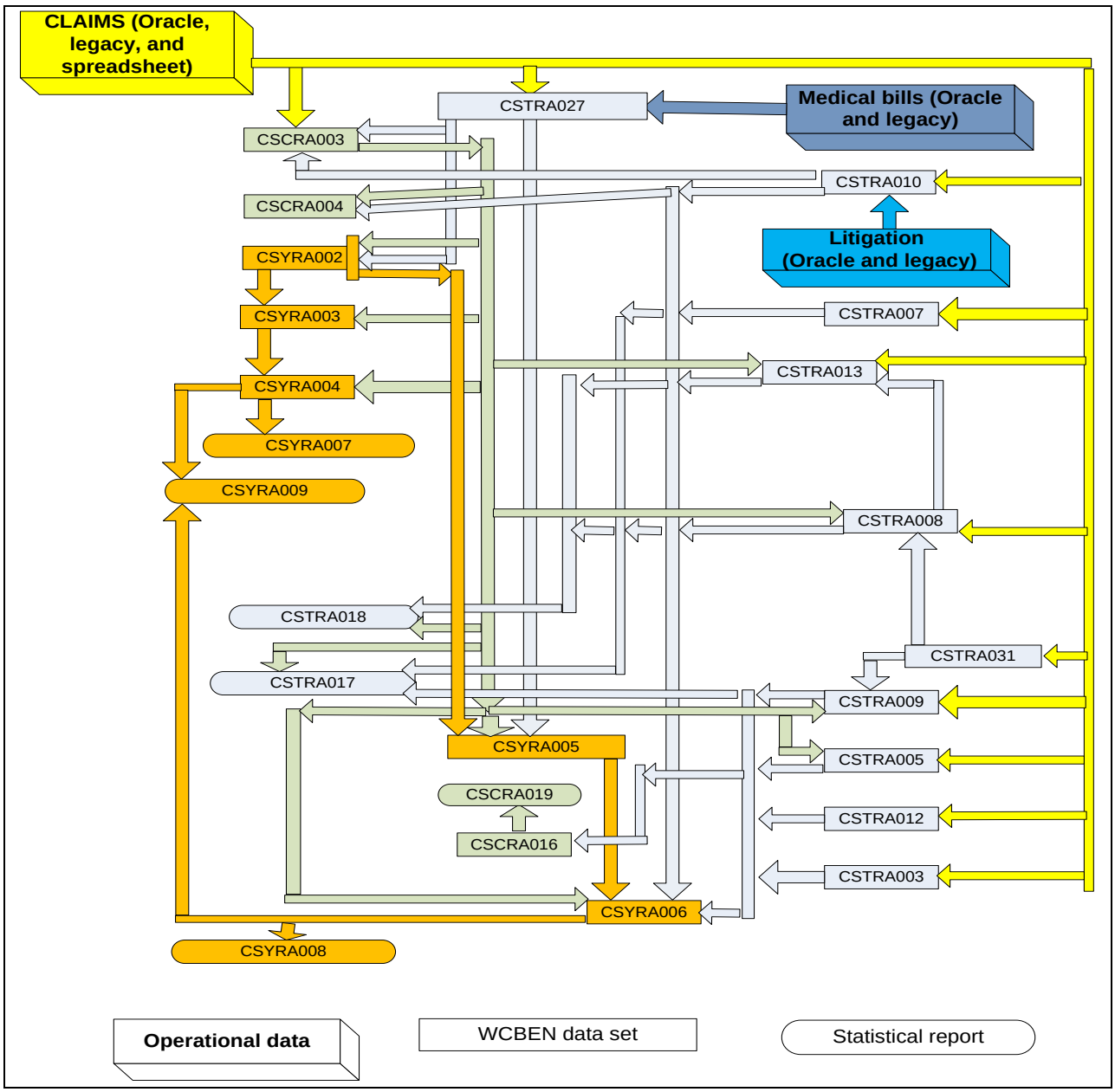

Figure 1. Simplified Extraction and Load for the WCBEN Data System, Excluding CSO Claim Outcomes

The programs may not be a model of efficiency, but the 13 that run in a production sequence every quarter require less than 30 minutes of wall time to load metadata and millions of claim and transaction records covering workplace injuries and illnesses since the mid-1980s. Production is in a scheduler, and each generation overwrites the previous,

though back-ups are available for a year. Appending records, an obvious way to simplify the extraction and load pathways, would be a larger task than is immediately apparent because of primary keys that may change and records that are sometimes deleted. Nevertheless, development of WCBEN programs has been an iterative process toward better understanding of more than 25 years of complex and changing data, business rules, data-entry, and data archiving practices. The level of that understanding shows itself in at least six facets of data preparation:

- A coherent structure for the data,
- minimal duplication of data,
- efficient SAS code,
- better data cleansing,
- useful documentation and metadata, and
- easy creation and validation of statistics.

Some of the advantages of working with Base SAS for good data preparation and successful application development are query capability through both the traditional DATA step and the SQL procedure; and built-in functions, formats, and procedures that replace many lines of code. Following are some highlights of extraction, transformation, and load operations done by the seven SAS programs that currently make up the CSO claim outcomes sub-system. Most of these are universally applicable, rather than UNIX-specific.

EXTRACTION

The goal of extraction is to get the rows and columns needed from their different data sources and formats and put them into a common structure. The most difficult extractions for the WCBEN system are in CSC and CST sub-system programs, from Oracle tables, flat files, and spreadsheets; but that would be the subject for another paper. Many of the resulting SAS data sets are the primary inputs for the CSO outcomes measures.

Each of the seven CSO programs has a well-defined and distinct purpose, including identification of outcomes-measure control and study claims (CSORA001), disability benefits for study claims (CSORA002 and 003), time-series benefits, employment, and wages for individual control and study claims (CSORA004 and 005), and summarized time-series data (CSORA006 and 007), which are in essence data sets of statistics.

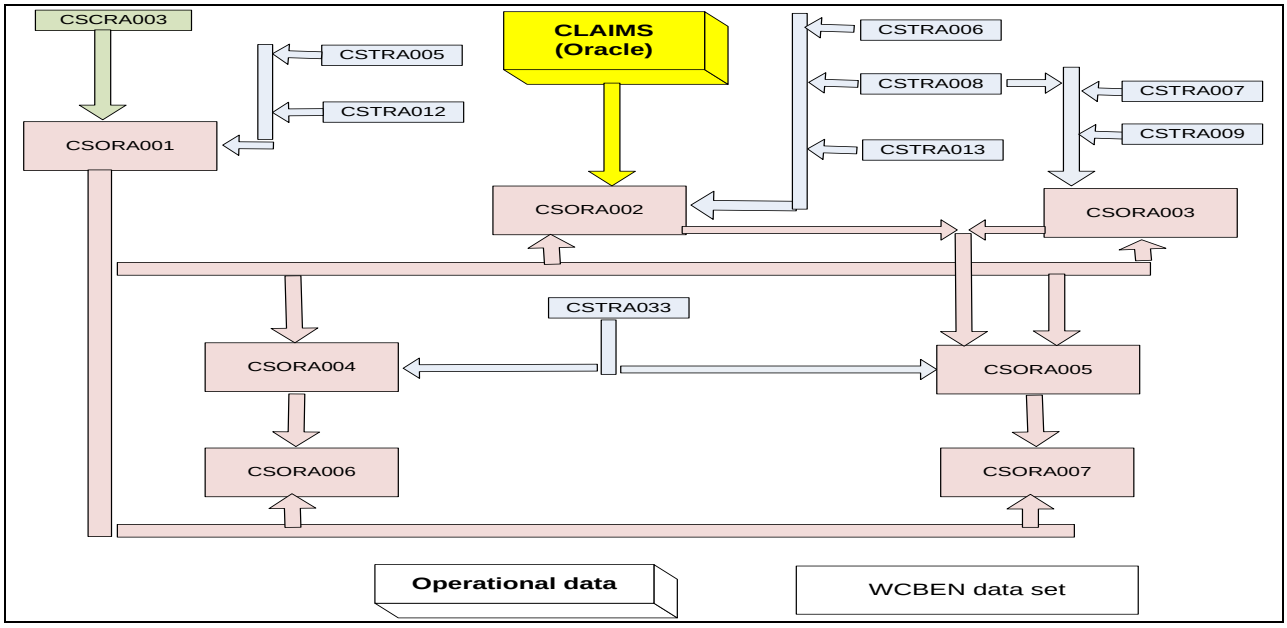

Figure 2. Simplified Extraction and Load for CSO Claim Outcomes

The CSORA001 claim record has identifiers that permit joins and match-merges to data about workers' compensation benefit payments and employment and wages, and it is part of the extraction for the remaining six CSO programs, as shown in Figure 2. Not by coincidence, the CSO load minimizes duplication of data, especially confidential identifiers, yet it is a much simpler pathway than the extraction and load of the underlying CSC and CST data shown in Figure 1.

FROM MANY RECORDS TO NOT SO MANY

The CSORA001 program has the most complex extraction, a subset of CSC claims records that meet criteria for assignment to the study and control groups. One claim per worker is allowed per fiscal year, but many workers have many claims, and some have more than one claim within a fiscal year. The extraction gives preference to study claims over control, to study claims where benefit payments have been reported to DCBS, and to the earliest such

record within a fiscal year. Central to this are two programming tasks: identification of workers with more than one claim record; and derivation of a hierarchical variable, *grp_order*, where the lower the value the higher the place in the ascending sort order for a worker's claim records and the higher the selection priority.

```
* 3. Create the study_class var, which splits the claims into study and control.
Create the grp_order var, which prioritizes the claims for inclusion in
the study claims, the controls, or exclusion from further analysis.
ID claims for workers with one and for workers with many. ;

proc format;
value $gord
    'ADC Closed'=1
    'ADC Init CDA'=2
    'ADC Reopen'=3
    'ADC Training'=4
    'ADC Open'=5
    'ANC EAIP'=6
    'ANC'=7;
*;
data acc_clm2 (drop=grp_flg);
merge acc_clm1 (in=inf1) EAIP_ATP2 (in=inf2);
by file_no;
if inf1=1;

if claim_grp='ANC' and inf2=0 then study_class='ANC-';
    else study_class='ADC+';

if grp_flg='E' and claim_grp='ANC' then grp_order=6;
    else if grp_flg='T' then grp_order=4;
    else grp_order=put(claim_grp, $gord.);
*;
Proc delete data=acc_clm1 EAIP_ATP2;
proc sort data=acc_clm2;
by worker_id;
data single_acc_clm1 mult_acc_clm1;
set acc_clm2;
by worker_id;
if first.worker_id and last.worker_id then output single_acc_clm1;
            /* unique claims per worker */
    else output mult_acc_clm1;      /* claims for workers with multiple claims */
run;
```

This SAS code, even if not the most elegant, illustrates implementation of business rules, attention to increased efficiency by decreasing use of CPU resources, and provision of documentation.

- Comments, enclosed by * and ; or /* and */ (and here though not in the program editor highlighted in yellow), don't execute but serve to explain the logic.
- The FORMAT procedure assists in the derivation of the *grp_order* variable, with the VALUE statement setting the business rule for priority.
- DATA statements are queries that create work data sets. The DROP= option deletes any variables no longer needed after data derivation and writing of the work data set.
- The SORT procedure orders the data going into both work data sets BY a desired variable or variables.
- The first work data set is created with a MERGE statement, of two data sets designated by IN= aliases *inf1* and *inf2* respectively. The match-merge is BY *file_no.* The IF *inf1*=1 statement assures that all the records from data set *inf1* will be written, along with any data from *inf2*, which previously had been stripped of duplicate *file_no* records according to more business rules.
- The first work data set creates the *study_class* variable that identifies study and control claims by IF … THEN and ELSE statements.
- It also derives the *grp_order* variable. Statements IF … THEN and ELSE … THEN convert single-character values from an *inf2* variable. The last statement in this routine, ELSE, uses the PUT function on multi-character values for an *inf1* variable, applying the character-to-numeric values from the PROC FORMAT. Note that numeric rather than alpha number values is merely adherence to the way that Oracle handles data.

- The DELETE procedure clears work space on the UNIX, a benefit to everyone who happens to be running SAS against the UNIX server at that moment.
- The second data step is a SET from the first, but it is preceded with PROC SORT to get a different BY variable, which is the reason for the new query.
- The second DATA step creates two work data sets. One is records for workers with single claims, from an IF statement that writes via the OUTPUT command a record that is both the FIRST and the LAST observation of the BY variable *worker_id*. All of those are loaded to the final CSO claims data set. The other work data set, *mult_acc_clm1*, is the records for workers with multiple claims, some of which are loaded.

The creation of the work variables FIRST. *worker_id* and LAST. *worker_id* by PROC SORT is a SAS functionality that is invaluable for common tasks of extraction, such as dealing with multiple records that need to be joined or merged to a single record when the objective is a single record. It's also a functionality that is difficult to reproduce in Oracle SQL. Here, however, the objective isn't a single record per worker, but a single record per worker within a grouping of records. The code that accomplishes that is interesting, but also long and involved, too much so for a detailed presentation in a paper already pressing the upper limits of allowed pages. Following, then, is a summary of the method chosen.

- The multiple-claims data set, *mult_acc_clm1*, is joined to itself via PROC SQL by *worker_id*, creating a table or work data set *mult_acc_clm_excl1* where the injury date for one claim (the appended) is greater than or equal to another (the base), with the base claim ID, injury date, and *grp_order* and the appended claim ID, injury date, and *grp_order* written to one to many base records as temporary or work variables. The author of this logic described it as "half a Cartesian product." For example, the query creates 120 records for a worker with 16 claim records.
- DATA *mult_acc_clm_excl2* SET *mult_acc_clm_excl1* identifies records for deletion where the appended injury date is within 366 days after the base injury date unless the appended *grp_order* has a higher-priority value. The resulting record is the claim ID from either the appended or base data. Working data *mult_acc_clm_excl2* is sorted by this variable, with the NODUPKEY option used to get rid of the duplicate records that are a consequence of a work data set from "half a Cartesian product."

Multiple-claim records to be kept for load to the final CSO data set result by subtracting the record marked for deletion from the original data set of multiple claims.

```
data mult_acc_clm2;
merge mult_acc_clm2 (in=inf1) mult_acc_clm_excl2 (in=inf2);
by claim_id;
if inf1=1 and inf2=0;
```

The code is similar to DATA *acc_clm2* above, with the crucial difference in the IF statement: the records from the *inf1* data set are written unless they are in the *inf2* data set.

STRANGE DATES

The second and third CSO programs extract from CST transactions the disability benefits paid for the study claims. Although the five input data sets undergo extensive data-cleansing routines prior to load, the author of CSORA002 discovered a new way to edit some of the data. The methodology was improved so that the logic relies upon operational data about maximum payment amounts, but with date fields having numeric year-month values, such as 200107, where the day is understood as either the first or last of the month, depending upon the field's definition.

These "strange" dates—undoubtedly there was a good reason to load them to Oracle in this way—are extracted and transformed to the SAS-date values characteristic of all the date variables in the WCBEN data system. A SAS date is a sequential-day date value where 0 represents 01Jan1960, and here it is useful for comparing values—the time sequence—from different date variables to determine whether a paid amount is above the maximum. Following is one way to make the transformation.

```
dby = int(doi_from / 100);
dbm = mod(doi_from, 100);
doi_from_dt = mdy(dbm,1,dby);
drop doi_from dbm dby;
```

Suppose that an extracted record has the numeric value 200107 for the field *doi_from*, and that value is shorthand for 01Jul2001. The INT function returns the integer value 2001, rather than 2001.07, from the operation 200107 / 100, as a temporary year variable. The MOD function returns the remainder from that same operation, which is 7, as a temporary month variable. The MDY function converts the values of the two temporary variables and the understood day value, 1, to a single SAS-date variable with a value that represents 01Jul 2001. Use of the DROP statement at the end of the routine increases efficiency by deleting variables not needed after the logic processes.

A final note about coherent data structure is that moving this data edit from CSORA002 to the applicable CST programs puts the data cleansing at the level within the WCBEN system where it belongs, because the CST logic includes a means-imputation routine for estimating values that would otherwise be deleted as outliers. Moving the data cleansing to CST greatly simplifies the CSORA002 extraction, as well.

TRANSFORMATION

Often enough, extraction requires transformation of the source data, such as seemingly strange dates. Strictly speaking, transformation is the data manipulation that changes the extracted data to the form and format needed for the load of the final data. Almost always, new variables are derived, but sometimes, new records.

DERIVING SEQUENTIAL-QUARTER VARIABLES

A transformation common to all seven of the CSO outcomes programs is the sequential-quarter variable, where an integer value is assigned to a quarter's worth of date values based on a fixed point in time or relative to a specific event common to all observations considered. In the CSO sub-system, sequential fixed-quarter variables are assigned values where 1 is an event during the first quarter of 2000, 2 is an event during 2000Q2, etc. Table 2 depicts the formula for assigning these values.

Date	Year of the date…	Minus 2000 equals….	Multiplied by 4 equals…	Plus quarter of date…	Equals fixed sequential quarter values
30Sep1999	1999	-1	-4	3	-1
01Dec1999	1999	-1	-4	4	0
28Feb2000	2000	0	0	1	1
30Apr2013	2013	13	52	2	54

Table 2. Formula for Sequential Fixed-quarter Variables, Value 1 = 2000Q1

The next two lines of code put this formula into effect by transforming the two different kinds of date values extracted by the CSO programs to sequential fixed-quarter variables. While *var1* is from a SAS date, *var2* is from another "strange date:" a string of five characters for the year and quarter, such as 20001.

```
var1 = (YEAR(SAS_date)-2000)*4 + QTR(SAS_date);
var2 = (SUBSTR(string,1,4)*1)-2000)*4 + (SUBSTR(string,5,1)*1)
```

For *var1*, the SAS date value is first converted to a numeric for the year via the YEAR function and then to a numeric for the quarter via the QTR function. For *var2*, the first 4 positions of the character string are extracted via the SUBSTR function and converted by arithmetic operator to a numeric for a year, and the fifth position of the string is converted to a numeric for the quarter.

The most important sequential-quarter variable for the times-series data is the sequential relative-injury quarter, which permits comparison of outcomes statistics at specific quarters relative to the injury no matter what year and quarter the injury and the outcome happened.

Fiscal year	Sequential relative-injury quarter	Outcomes with SAS dates for these quarters…	Are assigned fixed sequential quarter values where 1=2000Q1…	For injuries with SAS dates for these quarters…	Which are assigned fixed sequential quarter values where 1=2000Q1
2001	1	2000Q4	4	2000Q3	3
2001	1	2001Q1	5	2000Q4	4
2001	1	2001Q2	6	2001Q1	5
2001	1	2001Q3	7	2001Q2	6
2003	1	2003Q3	15	2003Q2	14

Table 3. Creation of Sequential Relative-quarter Variable from Two Sequential Fixed-quarter Variables

Table 3 illustrates the creation of this variable where the value 1 represents the first quarter after injury. The derivation is by subtracting the sequential fixed-quarter variable for injury date from the sequential fixed-quarter variable for outcome date.

MANY RECORDS FROM ONE

In addition to their role in deriving the sequential relative-quarter variable, the sequential fixed-quarter variables serve as part of the primary key or record ID and for MERGE and JOIN operations. Two of these variables are also used to get around the collection by DCBS of some disability payment data at prescribed events rather than prescribed intervals. That is to say, the CSORA002 program transforms a transaction record that often represents more than one quarter of benefit payments to records of payments for each of the quarters implied by the original record. Quarterly records are created within a DATA statement by a do loop:

```
do i = bsqtr to esqtr;
psqtr = i;
output;
end;
```

This iterative logic always starts with a DO statement and finishes with an END statement. The number of iterations is controlled by an index variable herein named *i* (and often named *i* in programs by the mathematically inclined). In the following example three records are created from one:

- Original record has *begin_date* 15Feb2001. The value for corresponding sequential-quarter variable *bsqtr* is 5.
- Original record also has *end_date* 15Aug2001. The *esqtr* value is 7.
- Variable *i* is processed three times, once for each value from bsqtr=5 TO esqtr=7. Note that TO is a key word in the syntax that indicates point to point, inclusive. Variable *i* has the values 5, 6, and 7, successively.
- The OUTPUT statement writes a record for each iteration, including another sequential fixed-quarter variable *psqtr*, the payment quarter, which takes the value of the *i* variable at each iteration.

Table 4 shows the three records created by the do-loop processing, where the variables from the original record have been kept, their values written to each of the new records.

Begin_date (original record)	Bsqtr (original record)	End_date (original record)	Esqtr (original record)	Pay_amount (original record)	Psqtr (payment quarter, new record, from i = bsqtr to esqtr)
15Feb2001	5	15Aug2001	7	2000	5
15Feb2001	5	15Aug2001	7	2000	6
15Feb2001	5	15Aug2001	7	2000	7

Table 4. Three records from one

If the original record's numeric variable *pay_amount* has a value of 2000, how then is the new variable *qtr_pay_amount* calculated for the three records? A quick and dirty way is to insert the next line of code into the do loop, before the OUTPUT statement:

```
qtr_pay_amount = round(pay_amount / (esqtr+1 – bsqtr),.01);
```

This formula uses the ROUND function to control the number of decimal places: cents, but not mils and smaller fractions. It resolves as 2000 divided by 3 is 666.67, where 3 is the count of quarters derived from the values for the two sequential fixed-quarter variables on the original record: *esqtr* is 7 plus 1 becomes 8, from which is subtracted 5, the *bsqtr* value.

The quick and dirty formula works best when *begin_date* is the first day of the quarter and *end_date* is the last day. The test for these conditions is most easily accomplished with the INTNX function.

```
If begin_date=intnx('quarter', begin_date, 0) and
end_date=intnx('quarter', end_date, 0, 'end') then …
```

Both lines of this code calculate a SAS-date value based upon the chosen interval 'quarter' from a date variable that has been incremented 0 quarters. The first line has no optional alignment argument after the increment argument 0 and so defaults to 'beginning', returning the SAS date for the first day of the same quarter. The second line has alignment argument 'end' and so the SAS-date value returned is the end date of the quarter.

The quick and dirty formula also works when the *begin_date* and *end_date* are within the same quarter: when *bsqtr=esqtr*, where the do-loop logic outputs one record from one record. In simplified terms, the formula for this condition is *qtr_pay_amount=pay_amount*.

Unfortunately, “quick and dirty” is an unacceptable method for calculating *qtr_pay_amount* most of the time, because the accuracy of the outcomes statistics for a given quarter (see Table 1) is vulnerable to the systematic distortion of the formula. The three quarterly records from Table 4 are not equal: payment quarter 6 appears to be a full payment quarter, but the other two quarters are obviously partial. Payment quarter 5, the begin quarter, is derived from *begin_date* 15Feb2001, which is in the middle of that quarter; and payment quarter 7, the end, from *end_date* 15Aug2001, also in the middle. In fact, beginning and end quarters are almost always partial-payment quarters: the begin date is not the first day of a quarter, and the end date is not the last day of a quarter. The values for *qtr_pay_amount* should be closer to 500 for payment quarter (*psqtr*) 5, 1000 for *psqtr* 6, and 500 for *psqtr* 7; and that may be accomplished by the INTNX function used to calculate ratios from the data retained on each new record from the original.

Records where *psqtr* is equal to neither *bsqtr* nor *esqtr* are full-payment quarters, assigned a value of 1 for *ratio*. That is true for the Table 4 record *psqtr*=6. Otherwise, the record is probably a partial-payment quarter, with a value for *ratio* from comparing *begin_date* (or *end_date*) retained from the original record to the INTNX-function derived begin and end dates for that quarter. Both *begin_date* and *end_date* for the Table 4 records are halfway through their respective quarters, and so *ratio* would be 0.5 for both the *psqtr*=5 and *psqtr*=7 records.

From the same original-record data retained on each new record, a calculation of the total quarters of payment covered by the original record, *qtrs_pd*, may be made—in essence, a sum of the ratios, written to each record. This is a long formula, repeating the derivations of *ratio* for both *bsqtr* and *esqtr*, and adding to that the count of full-payment quarters, from *esqtr*-*bsqtr*-1. Here, the formula resolves as 0.5 + 0.5 +1 equals 2.0 as the value for *qtrs_pd*, which is written to each record. It’s then a simple matter to calculate the more accurate values for *qtr_pay_amount*.

```
qtr_pay_amount = round((ratio / qtrs_pd) * pay_amount,.01);
```

LOAD

The SAS code standardizes a quarterly load or writing of the WCBEN data and metadata to the UNIX data warehouse by applying business rules systematically. The load for the CSO data sets includes the classification variables needed for the performance measures. While it minimizes the storage of confidential data that would permit identification of individuals, the load also provides each CSO data set with identifier variables useful for joins and match merges when even more detailed data from the WCBEN or operational data are needed for analysis of segments of the study-group claims.

The load is a permanent integration of the disparate data needed for more or less complex extraction and transformation. The load advances the goal that different analysts will use the same data to get the same answer to a single analytical question, every time.

GENERATIONAL LOAD

Actually, not quite every time, for workers’ compensation claims data are by their nature a temporary snapshot of the activity affecting a claim. Whether a claim is still open and payments may be due, how much has been paid to date, and the like—these are subject to change as a claim develops. In Oregon, getting a handle on claim development, predicting or forecasting it, is complicated by the DCBS reporting requirement for benefit payments at prescribed events, which also is the reason for the transformation of one record to many that was just described. A generational load—where each CSO data set that is created each quarter is kept—is scheduled so that claim development data for imputing benefit payments are available to estimate outcomes for the substantial number of claims that haven’t developed to the point of reported benefits. The load is accomplished through PROC SQL.

```
Libname ben '/mypath/myfolder';
call symputx('date',compress(put(today(),mmddyy10.),'/'));
proc sql;
create table ben.myfile_&date as ...
```

The LIBNAME statement sets an alias *ben* for the path to the folder in which the data set is loaded. The CALL SYMPUTX routine is used with the PUT, COMPRESS, and TODAY() functions to create a macro variable *date* with the value of today’s date in the specified format with the special character / removed: for example, 30Apr2013 becomes 04302013. The CREATE statement in the PROC SQL uses these to load a data set, for example myfile_04302013, to the correct place in the data warehouse. The addition of a date to the data set name, by invoking the macro variable name with the prefixed symbol *&*, distinguishes the quarterly generations.

DOWNLOAD TO SPREADSHEET

Optionally, data sets may be loaded to spreadsheet for further analysis and graphical depiction. This has been useful for the time-series statistical data loaded by the sixth and seventh CSO programs. The SAS Output Delivery System

(ODS) is an immense topic, but the next few lines of basic code produce an html that is downloaded from a temporary file on the UNIX to the desired folder on the LAN. When saved as a true spreadsheet file, these data may be manipulated according to all the functionality of Microsoft Excel.

```
filename tab1 '/mypath/myfolder/myfile.xls';
ods html body=tab1 style=minimal;
proc print …
ods html close;
proc download infile='/mypath/myfolder/myfile.xls'
outfile='/mypath/myfolder/myfile.xls';
```

DATA VALIDATION, DOCUMENTATION, AND METADATA

The SAS programs also produce metadata: the data system documentation that is vital for comprehension of the WCBEN system by other analysts. Metadata is embedded in the code as attributes for each variable, and through the ability to write comments that document the logic but aren't processed by SAS. The comments explain what each section of code is meant to accomplish.

The processing of the code results in metadata in the SAS Log: an example is the SAS Note that returns a count of observations (records) and number of variables (fields, columns) produced by a query. These notes are useful for debugging, especially in those all too common situations where the log returns no ERROR message but results are otherwise unexpected.

The label attribute for a variable is optionally set by a LABEL statement during the DATA step or a LABEL option in the SELECT statement of PROC SQL. It's always applied for WCBEN loads because it provides a description that is more meaningful than the variable name, which most often is a shorthand name applied to a derived variable.

Variable name	Label
dsqtr	Sequential relative-injury quarter
pct_wage	Wages, percent base wages
pct_indem	Indemnity, percent base wages

Table 5. Labels Are Metadata for Variables

Metadata are also created in output from SAS procedures. The LABEL or the variable name may be used in the output depending upon the purpose, and many lines of explanatory text may be added through TITLE and FOOTNOTE statements.

- The CONTENTS procedure includes the data set name, location, and number of records; and column (variable) position, name, data type, length, optional formats, and optional label.
- The FREQ procedure is an easy way to get distributions of classification variables' values.
- The UNIVARIATE, MEANS, or TABULATE procedure provides statistics about benefits payments, for example, which assist in validating the SAS program's logic.
- PROC PRINT yields a list, typically the first 100 records in a data set or a more complex sample of records, from which the data may be better understood, and problems in the transformation may be detected.

These and other procedures are run against all WCBEN data sets. They may of course also be run against work data sets when de-bugging a SAS program or otherwise verifying that extraction and transformation have been done correctly.

```
Proc sort data=dat1 nodupkey out=dat1_samp;
By var1 var2;
Proc print data= dat1_samp;
```

This bit of SAS code uses the SORT procedure's NODUPKEY and OUT option to provide a complex sample of records, one for each combination of the classifiers *var1* and *var2.* If both variables have 8 values, then there will be a sample list of 64 records out of let's say a million records from the work data set dat1.

CONCLUSION

Base SAS is used in a UNIX environment to gain knowledge of disparate and scattered data for workers' compensation costs paid and employment and wages. It provides ETL capability for application development, a system of metadata and SAS data sets of worker benefit costs and outcomes statistics, useful for performance measurement and many other reporting needs.

REFERENCES

- Hunt, Allan, et al. April 2006. “Earnings Losses for Injured Worker.” *Employment Research*. Kalamazoo, MI: W.E. Upjohn Institute for Employment Research.
- Oregon Department of Consumer and Business Services. February 2013. *Oregon Workers’ Compensation Premium Rate Ranking Calendar Year 2012*.
- Reville, Robert, et al. 2001. *An Evaluation of New Mexico Workers’ Compensation Permanent Partial Disability and Return to Work*. Santa Monica, CA: RAND Institute for Civil Justice.
- Siegel, Eric. 2012. *Predictive Analytics Applied*. San Francisco: Prediction Impact, Inc.
- Welch, Edward. 2000. *Final Report: Oregon Major Contributing Cause Study*. Lansing, MI: Michigan State University.

ACKNOWLEDGMENTS

Gary Helmer, Economist, is the project leader for conceptual design and oversight, original author and steward of the outcomes data and statistics, and reviewer of the follow-up work done by the author of this paper. Ronni Rachele, Research Manager, and Mike Manley, Research Analyst, are the primary advisors. John Glen, Research Analyst, Oregon Employment Department, annually provides an extract of employment and wage data so that DCBS may construct performance measurements.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Mike Maier
Department of Consumer and Business Services / Central Services Division
P.O. Box 14480
Salem, OR 97309-0405
Work Phone: 503-947-7352
Fax: 503-947-7085
E-mail: Mike.g.maier@state.or.us
Web: http://www.oregon.gov/DCBS