Paper 111-29

Developing Data Marts and Web-Enabled OLAP for Actuarial and Underwriting Analysis

By Michael Krumenaker – Sr. Project Manager, Palisades Research, Inc.
George Bukhbinder – President, Palisades Research, Inc.

Abstract

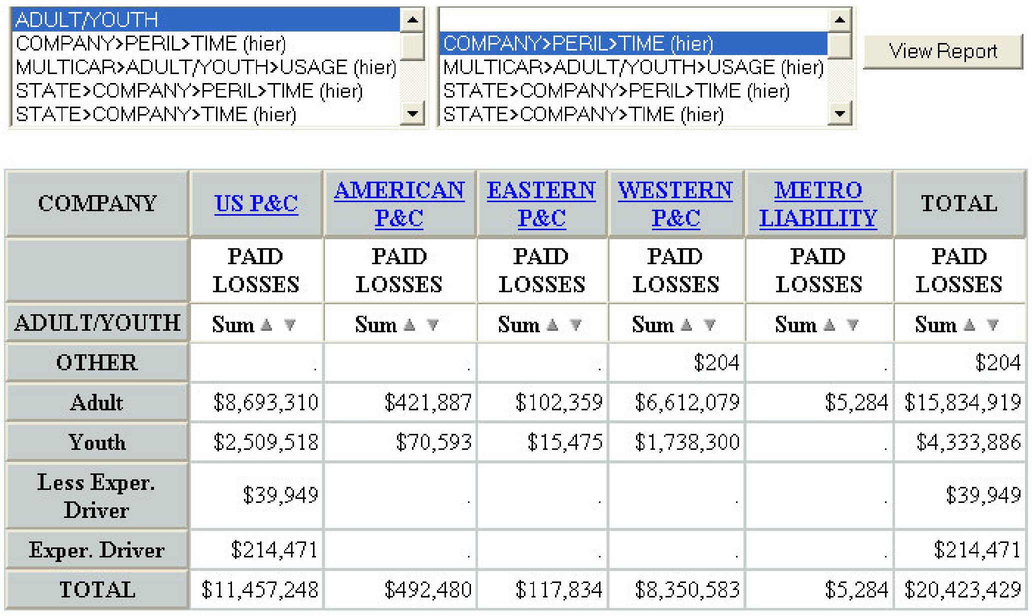
A data warehouse or data mart is the foundation of an on-line analytical processing (OLAP) system used to develop pricing models in the insurance industry. The issues we face in creating such a data mart and multi-dimensional database (MDDB) for actuarial and underwriting departments include (1) which factors to evaluate using OLAP, i.e., selecting the class variables - e.g., age, experience, points - and analysis variables related to premiums and claims, (2) how to most efficiently and accurately develop MDDB cubes from customer transaction records in the data warehouse, and (3) how to empower business users to change business rules without requiring changes to hard code, using popular user-friendly software and a simple syntax. The overall solution must achieve a variety of goals: Minimizing processing time requires efficient code; ensuring accuracy with routines to detect data anomalies throughout the process; maximizing the solution's analytical value, which requires close consultation with the users and an understanding of their business issues. We can enable business users to manage changes in certain business rules without re-programming by using tools familiar to users (e.g., Microsoft Excel) and the SAS® macro language to automatically generate new code during each periodic production run.

Choosing class and analysis variables for each MDDB cube

Analysis variables are the quantities being measured. The analysis variables include premium, the number of claims, the amount of loss incurred and the amount paid, and allocated expenses. Class variables are variables whose effects are being measured individually or in concert with other class variables. For auto insurance, class variables include the type of coverage ("peril"), geographic location (region, state and/or other levels), the number of times renewed ("renewals"), insurance company "points", coverage limits, deductibles, driver training, measurements of age and driving experience, and the time period – month, quarter, or year, as the case may be. The class variables in each cube are presented in hierarchical fashion, and different hierarchies can be presented for the same combination of variables. For instance, if the class variables in a particular cube are state, quarter, peril, renewals, policy limit, and age, the same cube can accommodate the hierarchies *renewals-deductible-age* and *deductible-age-renewals*.

Software such as the SAS MDDB viewer allows the user to select dimensions (class variables) both across and down on screen and facilitates drill-down through the hierarchies. The MDDB viewer allows the user to select multiple class variables in the across and/or down dimensions, in which case all permutations of the selected dimensions appear on-screen at once, or to select hierarchies as illustrated in figures 1 and 2.

The MDDB cube contains the results of cross-tabularization to measure the effect of various combinations of class variables, at various levels of aggregation, on each of the analysis variables. The number of possible combinations of class variable values can be enormous, depending on the number of class variables and aggregation levels. The larger the cube, the longer it takes to retrieve the information. Therefore it is usually undesirable to create a single cube containing all the class variables. The users probably do not need or even want a cube that would permit them to evaluate every possible combination of class variables. The users' analytical requirements determine the class and analytical variables in the MDDB cube or cubes, so that the users determine which class variables are required and in what combinations. This is a business decision. But while accommodating the selected class variable combinations, the system architect determines how the class variables should be aggregated. For instance, if the user specifies that the system present the hierarchies *renewals-age-deductible*, *renewals-mileage* and *usage-renewals-mileage*, the system architect determines whether this calls for the creation of one, two or even three separate cubes, with renewals appearing in each cube, based on considerations of request-processing time (favoring smaller cubes), the file size of each cube, and the time required each month to create each cube. It may even be optimal to create one cube for each combination of class variables the user needs. Each such cube can accommodate all drill-down and hierarchical arrangements for the class variables it contains, i.e., a single cube can accommodate the hierarchies `var1-var2-var3 and var3-var1-var2`.



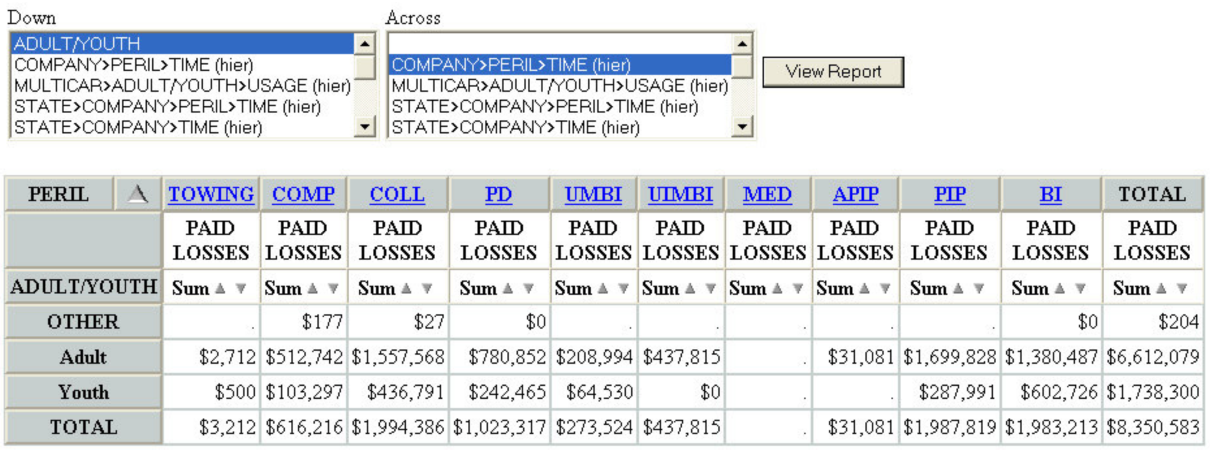
ADULT/YOUTH
COMPANY>PERIL>TIME (hier)
MULTICAR>ADULT/YOUTH>USAGE (hier)
STATE>COMPANY>PERIL>TIME (hier)
STATE>COMPANY>TIME (hier)

COMPANY>PERIL>TIME (hier)
MULTICAR>ADULT/YOUTH>USAGE (hier)
STATE>COMPANY>PERIL>TIME (hier)
STATE>COMPANY>TIME (hier)

View Report

COMPANY	US P&C	AMERICAN P&C	EASTERN P&C	WESTERN P&C	METRO LIABILITY	TOTAL
	PAID LOSSES	PAID LOSSES	PAID LOSSES	PAID LOSSES	PAID LOSSES	PAID LOSSES
ADULT/YOUTH	Sum ▲ ▼	Sum ▲ ▼	Sum ▲ ▼	Sum ▲ ▼	Sum ▲ ▼	Sum ▲ ▼
OTHER	.	.	.	$204	.	$204
Adult	$8,693,310	$421,887	$102,359	$6,612,079	$5,284	$15,834,919
Youth	$2,509,518	$70,593	$15,475	$1,738,300	.	$4,333,886
Less Exper. Driver	$39,949	.	.	.	.	$39,949
Exper. Driver	$214,471	.	.	.	.	$214,471
TOTAL	$11,457,248	$492,480	$117,834	$8,350,583	$5,284	$20,423,429

Figure 1: MDDB view in browser before drill-down



Down
ADULT/YOUTH
COMPANY>PERIL>TIME (hier)
MULTICAR>ADULT/YOUTH>USAGE (hier)
STATE>COMPANY>PERIL>TIME (hier)
STATE>COMPANY>TIME (hier)

Across
COMPANY>PERIL>TIME (hier)
MULTICAR>ADULT/YOUTH>USAGE (hier)
STATE>COMPANY>PERIL>TIME (hier)
STATE>COMPANY>TIME (hier)

View Report

PERIL ▲	TOWING	COMP	COLL	PD	UMBI	UIMBI	MED	APIP	PIP	BI	TOTAL
	PAID LOSSES	PAID LOSSES	PAID LOSSES	PAID LOSSES	PAID LOSSES	PAID LOSSES	PAID LOSSES	PAID LOSSES	PAID LOSSES	PAID LOSSES	PAID LOSSES
ADULT/YOUTH	Sum ▲ ▼	Sum ▲ ▼	Sum ▲ ▼	Sum ▲ ▼	Sum ▲ ▼	Sum ▲ ▼	Sum ▲ ▼	Sum ▲ ▼	Sum ▲ ▼	Sum ▲ ▼	Sum ▲ ▼
OTHER	.	$177	$27	$0	.	.	.	.	.	$0	$204
Adult	$2,712	$512,742	$1,557,568	$780,852	$208,994	$437,815	.	$31,081	$1,699,828	$1,380,487	$6,612,079
Youth	$500	$103,297	$436,791	$242,465	$64,530	$0	.	.	$287,991	$602,726	$1,738,300
TOTAL	$3,212	$616,216	$1,994,386	$1,023,317	$273,524	$437,815	.	$31,081	$1,987,819	$1,983,213	$8,350,583

Figure 2: MDDB view in browser: Drill-down through COMPANY Western P&C (total paid losses = $8,350,583)

Developing class and analysis variables from records in the data warehouse

Data for the mart may be obtained from a data warehouse containing transactional records. Such records must be pre-processed and summarized before being used to create MDDB cubes. Preprocessing may include selecting relevant fields from customer records, using lookup tables to add additional information to each record, e.g., limitations which vary from state to state or the values of certain variables based on rate class codes (see section 3 of this paper).

The entire process is summarized in Figure 3. In a data warehouse comprised of transactional data, one record is created each month for each customer to record earned premium, and one or more additional premium records are created whenever one or more policy characteristics (e.g., persons covered, limits, deductible, perils – class variables) change or the policy is renewed. If policy characteristics do not change, then part of the current month's record is redundant with the previous month's policy characteristic information. To eliminate this redundancy, for the data mart these records are processed into quarterly customer records that hold the repeating policy characteristic information ("policy history") in a dataset separate from the monthly premium record and claims information, in which each record includes the start and end date to which it applies. For example, if the characteristics of a policy do not change for four months, then four policy history records are collapsed into one, saving storage space and, more importantly, speeding up processing. The policy history dataset is in the second row of Figure 3.

A detailed discussion of processing the customer records into MDDB cubes is beyond the scope of this paper.[1] To briefly summarize: In the architecture we employed to create an insurance data mart, transactional data in the insurance company's data warehouse is on the mainframe. Each month, it is pre-processed in part on the mainframe using SAS (Figure 3, first row – monthly files). The resulting monthly files are further pre-processed by SAS on a Unix server (Figure 3, second row), in data steps and procedure that include the determination of the values of rate class code variables (see section 3 of this paper), and the separation of premium data (maintained on a monthly basis until quarterly summarization) from policy history, as explained in the previous paragraph. At the end of each quarter, the monthly files (claims, premium) are summarized into quarterly files (Figure 3, third row). These quarterly summaries are available for ad hoc queries. Finally, the quarterly summary files are processed into MDDBs, each covering 24 quarters. To the extent possible, MDDBs are revised rather than fully reconstructed.[2]

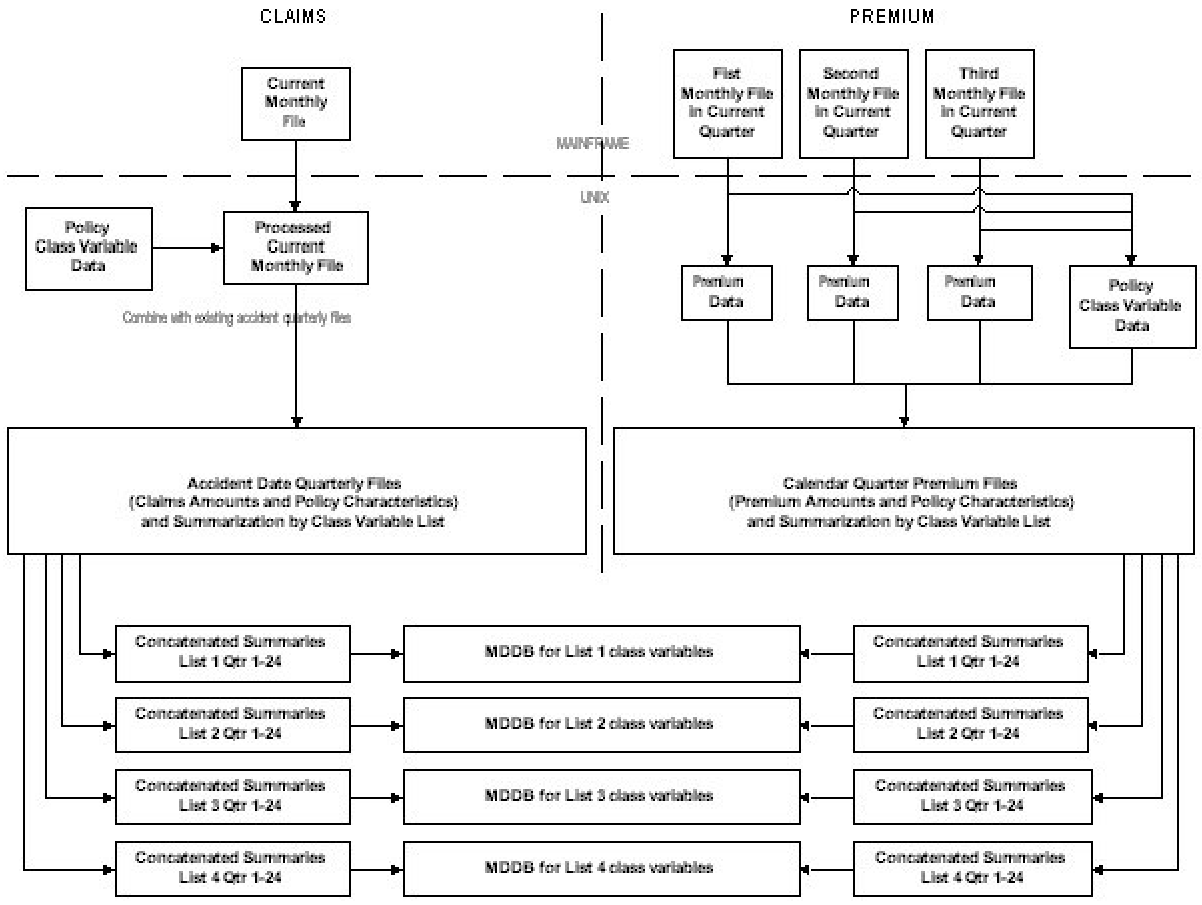

CLAIMS
PREMIUM
Current Monthly File
First Monthly File in Current Quarter
Second Monthly File in Current Quarter
Third Monthly File in Current Quarter
MAINFRAME
UNIX
Policy Class Variable Data
Processed Current Monthly File
Premium Data
Premium Data
Premium Data
Policy Class Variable Data
Combine with existing accident quarterly files
Accident Date Quarterly Files (Claims Amounts and Policy Characteristics) and Summarization by Class Variable List
Calendar Quarter Premium Files (Premium Amounts and Policy Characteristics) and Summarization by Class Variable List
Concatenated Summaries List 1 Qtr 1-24
MDDB for List 1 class variables
Concatenated Summaries List 1 Qtr 1-24
Concatenated Summaries List 2 Qtr 1-24
MDDB for List 2 class variables
Concatenated Summaries List 2 Qtr 1-24
Concatenated Summaries List 3 Qtr 1-24
MDDB for List 3 class variables
Concatenated Summaries List 3 Qtr 1-24
Concatenated Summaries List 4 Qtr 1-24
MDDB for List 4 class variables
Concatenated Summaries List 4 Qtr 1-24

Figure 3: Creating Insurance Industry MDDBs from Customer Data

Dealing with millions of records each month, requires both efficient architecture and efficient code. Input/output is one of the biggest uses of real running time even if the data step or procedure involved uses little processing time, so (1) minimize the number of passes through the records (the number of data steps and procs), and (2) reduce record length for processing by dividing each record into two parts, a "skinny" part containing fields which will be processed (sorted, summarized, etc.), and the other containing fields that remain static. After processing, the skinny part can be merged back with the static part.

Implementing business rules changes without code changes

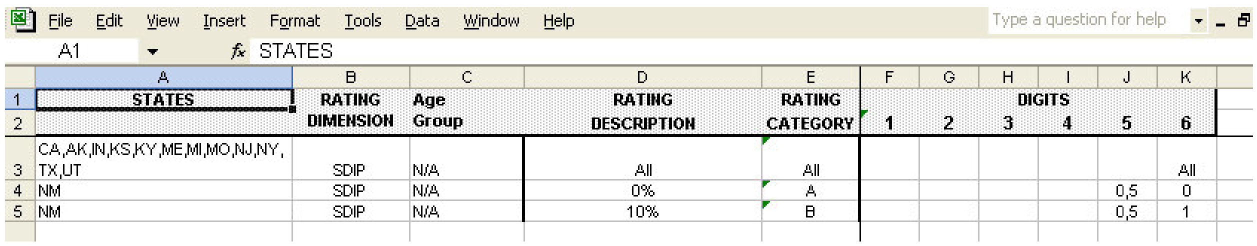
The value of ten class variables used in policy rating (pricing) are encoded into a short alphanumeric text string (mostly numeric) called the "rate class code". Here, "class" does not refer to the class variables in the MDDBs. The variables in question are called "rate class code variables," also called "rating dimensions". Examples of rate class code variables are age group, vehicle usage, and insurance "points." Since insurance is a state-regulated business, the business rules governing the rate class codes vary by state, and any given state's premium rating methodology involves hundreds of different rate class codes. Furthermore, the structure of rate class codes varies substantially by state. Therefore, the program code to interpret the rate class code in conjunction with the State and other factors may entail pages and pages of pure complexity. Changing this code requires a programmer and thorough testing after re-coding.

Here is another way. The user enters changes to the business rules for interpreting rate class codes through a familiar front end, such as Microsoft Excel. Use SAS macros to interpret the contents of the Excel workbook and as the calculation engine. The SAS program (the "permanent code") must be flexible enough to allow the user to easily add or change parameters, formats and business rules, and even to add new variables, through the familiar user interface without requiring any changes in the production program.

The concept of separating business rules from code is more important than the actual techniques involved. Flexible code that can accommodate changes in business rules that the client can input easily adds significantly to the value of the application. An in-depth discussion of the macro coding and program flow-of-control are beyond the scope of this paper,[3] but we will go over the basic techniques.

Figure 4 shows a portion of one of two "rules" worksheets in one such workbook interface. The real values have been disguised. The value of **`SDIP`** (insurance points) is based on an interpretation of the six-digit rate class code (columns F-K) in conjunction with the **`State`** (column A). **`Age Group`** is not a factor in determining **`SDIP`**, but is a factor for other rate class code variables, the rules for which appear in subsequent rows (not shown here). Column D contains the formatted version of the value of the class code variable identified in column B, and column E contains the value for that variable that is read into the SAS dataset. Row 3 contains the following business rule: In **`States`** CA, AK,…, UT, for any value in the sixth character (we will say "**`digit`**") of the class code the value of **`SDIP`** is the same as that **`digit`** (hence "All" in columns D and K). Line 4 says that in New Mexico, if **`digit5`** is 0 or 5, and **`digit6`** is 0, then **`SDIP`** = 0% (and the value "A" goes into the dataset). The rule in line 6 is the same as line 5 except that **`digit6`** is 1.

The macro code "explodes" multi-state records such as row 3 to produce one record for each state. For each of the 12 records derived from line 3, the macro code then "explodes" the "All" into all possible values (0-9, A-Z) (total 12 x 36 = 432 records), and similarly operates on the pair (0,1) for the record in lines 4 and 5. So these three records are ultimately "exploded" into 436 (432 + 2 + 2) different rules.



	A	B	C	D	E	F	G	H	I	J	K
1	STATES	RATING	Age	RATING	RATING			DIGITS			
2		DIMENSION	Group	DESCRIPTION	CATEGORY	1	2	3	4	5	6
3	CA,AK,IN,KS,KY,ME,MI,MO,NJ,NY, TX,UT	SDIP	N/A	All	All						All
4	NM	SDIP	N/A	0%	A					0,5	0
5	NM	SDIP	N/A	10%	B					0,5	1

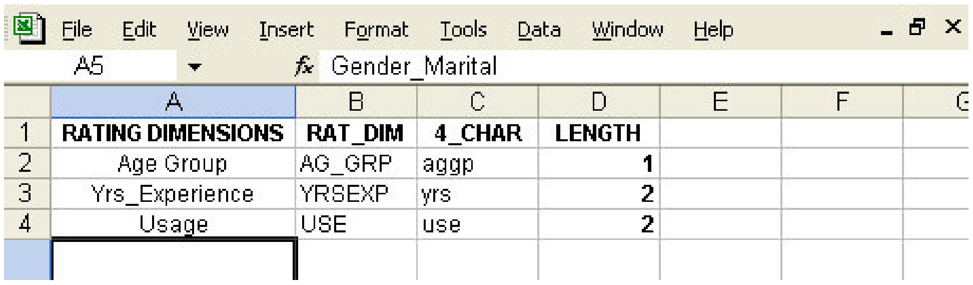
Figure 4: A Portion of the Rules Definition Sheet in Rate Class Code Interface Workbook

The SAS macros need other information about each class code variable in order to create the interpretive data steps and procedures each month:

1. A descriptive name (for use as LABELs, assigned by ATTRIB statements)
2. A variable name for use by SAS
3. A short name (say 4 characters)
4. Length

5. A list of allowable values (more for Excel than for SAS)
6. Formats

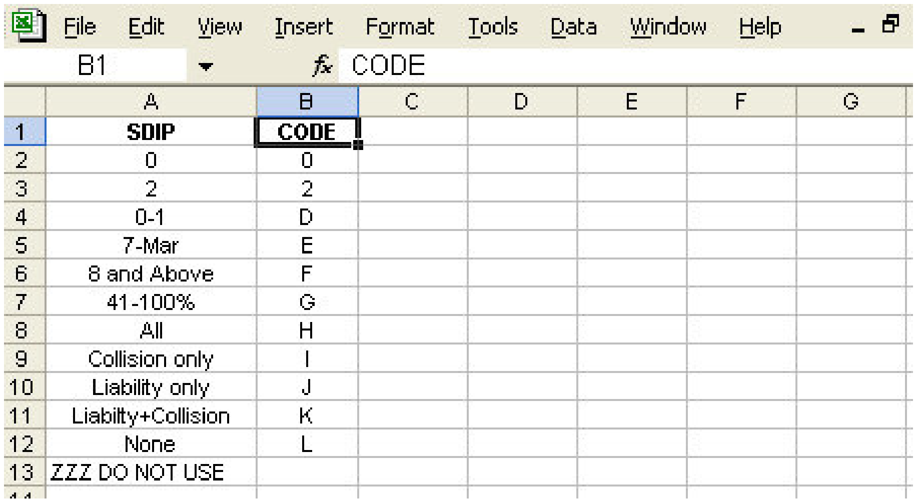
Items 1-4 are set forth in a small workbook page having one line for each rate class code variable. If there are ten rate class code variables, they are completely described on ten lines. Figure 5 shows three such lines.



	A	B	C	D
1	RATING DIMENSIONS	RAT_DIM	4_CHAR	LENGTH
2	Age Group	AG_GRP	aggp	1
3	Yrs_Experience	YRSEXP	yrs	2
4	Usage	USE	use	2

Figure 5: A Portion of the rating dimension worksheet

The list of allowable values and their formats are set forth in an individual worksheet for each rate class code variable. Such a worksheet for SDIP might appear as in Figure 6. Values are in column B, and formats are in column A.



	A	B
1	SDIP	CODE
2	0	0
3	2	2
4	0-1	D
5	7-Mar	E
6	8 and Above	F
7	41-100%	G
8	All	H
9	Collision only	I
10	Liability only	J
11	Liabilty+Collision	K
12	None	L
13	ZZZ DO NOT USE	

Figure 6: Worksheet Page with Allowable Values and Formats for SDIP

The value of each of the ten class code variables is determined by up to nine factors: **State**, **Company**, **Age Group**, and the value of one or more of the six **digits** which comprise the rate class code. **State** is always relevant, but each of the other eight factors may or may not be relevant. In row 3 in Figure 4, neither **Company** nor **Age Group** is relevant, and only digit 6 of the class code is relevant. If **Age Group**, **company**, and each **digit** is represented by one bit in a single byte, and "1" means that the factor is relevant while "0" means it is not, then there are 255 (2^8 – 1) possible relevance patterns (exclude 00000000 - no relevant factors). We represent each possible relevance pattern for each of the ten rate class code variables by a global macro variable, each of which initialized to 0. We create the global macro variables **&use1**, **&use2**,…, **&use255**, **&mileage1**, **&mileage2**, etc., for a total of 2,550 (255 x 10 variables) global macro variables ("flags", all initialized to 0). Then, as the macros process the rules worksheets, they determine which relevance combinations are actually used and "flip" the value of the corresponding flags to 1. For example, the relevance pattern in row 3 in Figure 4 causes flag **&sdip1** (00000001=1) to be flipped to 1 (meaning "yes").

How do we use the 2,550 flags? Each flag that was flipped causes the program to create a data step that in turn creates a data set with the records from the exploded rules worksheets matching the corresponding relevance pattern. How? We use another set of 255 global macro variables (named **&bin1, &bin2,..., &bin255)** to provide a picture of each relevance pattern. These globals have literal string values '00000001', '00000010',…'11111111' corresponding to their numeric designation. For example, **&bin16** = '00010000'.

We test each of the 2,550 flags via two nested do loops. Since **&sdip1** = 1, the macro code dynamically creates a data step that puts 432 records into a data set named **sdip1**. The program macros specify that this data set will contain only the relevant variables (in this case, **digit6**) by referring to the string patterns in the corresponding **&bin** variable. For example, for the data set created because the flag **&sdip1** = 1, the program looks at the pattern in **&bin1** (00000001) and puts **digit6** (always in conjunction with **State**) and no other variable in the data set **sdip1**, which will have one record for each **State** to which the rule it embodies applies.

The program then uses the data sets such as **sdip1** as lookup tables in processing the customer records. The program macros create a single data step which includes multiple set statements – one for the customer records data set, and one for each of the lookup table datasets created based on the relevance patterns. The resulting data step calls a macro **%do_sets** for each of the ten rate class code variables. Again looking at the value of each of the 2,550 flags, a macro automatically adds the necessary set statements to the data step to use the relevant lookup tables. Parameters for **%do_sets** are the variable name (e.g., **sdip**) and the set name (e.g., **sdip1**). Each set name is represented by **&&&setname&i**, e.g., if i = 1 then **&&&setname&i** resolves to **sdip1**. The set name pulls double duty, since it is also the macro variable name for the corresponding flag. Thus, **%do_sets** generates the data step that uses the macro-generated lookup tables (data sets) to evaluate the rate class codes:

```
%macro do_sets(var,setname);
    %let marker = 0;
    %do i = 255 %to 1 %by -1;            *iterate downward;
        %if &&&setname&i = 1 %then %do; *&&&setname&i: the flag macro variable;
            %*marker is 1 if there was a previous hit for this variable.;
            %if &marker = 1 %then %do;
                if &var = "" then do;
                    set &setname&i key = X%upcase(&setname)&i / UNIQUE;
                    if _IORC_ <> 0 then _ERROR_ = 0;
                end;
            %end;
            %else %do;
                set &setname&i key = X%upcase(&setname)&i / UNIQUE;
                if _IORC_ <> 0 then _ERROR_ = 0;
                %let marker = 1;     *Prevent reprocessing of records.;
            %end;
        %end;    %*if setname_i = 1...;
    %end;        %*do i = 255 to 1 by -1;
%mend;
```

Conclusion

Turning customer records into MDDB cubes suitable for decision support requires efficient data mart and cube architectures, close consultation with the business users concerning the desired combinations of class variables and analytical variables, and every programming efficiency you can bring to bear. Subsequent re-programming can be avoided to the extent that business rules can be represented by a simple syntax and using a tool familiar to the users if SAS macros are used instead of hard coding for such business rules.

CONTACT INFORMATION

George Bukhbinder President Palisades Research, Inc. 75 Claremont Road, Suite 312 Bernardsville, NJ 07924 908-953-8081 mkrumenaker@palisadesresearch.com	Michael Krumenaker Sr. Project Manager Palisades Research, Inc. 75 Claremont Road, Suite 312 Bernardsville, NJ 07924 908-953-8081 mkrumenaker@palisadesresearch.com

References

[1] See G. Bukhbinder and M. Krumenaker, Internet-Enabled Information Delivery from Data Warehouses for the Telecommunications and Insurance Industries, Proceedings of the 15th Northeast SAS Users Group (Sept. 2002).
[2] See Bukhbinder & Krumenaker, section 5 under Data Model Design.
[3] See M. Krumenaker and J. Bhattacharya, User Implementation and Revision of Business Rules Without Hard Coding: Macro-Generated SAS Code, Proceedings of the Northeast SAS Users Group, Sept. 2003.