Healthcare Intelligence: The Challenge of OLAP for Healthcare Data

John Leveille,
d-Wise Technologies
Raleigh, NC

www.d-wise.com

ABSTRACT

Some data sources easily lend themselves to aggregation within OLAP cubes. In contrast, when attempting to apply OLAP technology to healthcare and pharmaceutical data sources, even the simplest set of data can leave you stymied. This paper demonstrates how to use SAS® Business Intelligence and PROC OLAP to create two challenging cubes: a per member per month cube from insurance claim data and a quality of care cube from patient chart data.

Introduction

The past decade has seen a steady increase in the adoption of tools to explore aggregated data. During this time the software marketplace has grown to contain many vendors offering similar cube and pivot table technologies. This growth in the software market is naturally correlated with the alignment of business operations, across corporations and across industries, applying these technologies under the guidance of conventional wisdom and standard strategies. As is often the case, the computer solves the easy problems quite well. We can see this with the application of business intelligence cubes, or Online Analytical Processing (OLAP) technology, to routine business operations.

Here income, expenses, inventory, sales are easily added up across lines of business, regions and other groupings. Obviously, there are challenges within this setting, but it presents many simple aggregations that are easily solved by the stock tool set. In contrast, when attempting to apply OLAP technology to healthcare and pharmaceutical data sources, even the simplest set of data can leave you stymied. How does one add the blood pressure for two patients? What does it mean even if you can aggregate these data? How can you roll up per member per month insurance calculations, adding the dollars spent, but keeping the member count constant? This paper demonstrates how to use SAS Business Intelligence and PROC OLAP to create a few challenging cubes an enable Healthcare Intelligence.

Determine Your Aggregations And Measures

Before you begin to build an OLAP cube you need to understand your data well. The end result of this activity is to produce a tool for decision makers and analysts. They will be using the cube that you build to gain insights into and across what is likely a diverse population of individuals. Once you understand the source data from which you will build one or more cubes, you should determine how you would like to aggregate the data. Since many healthcare cubes are composed from complex source data, it is best to initially focus on a single aggregation and a single measure in each cube. After you have succeeded in building a cube and verified its correctness, you can rebuild it with additional aggregations and measures to enhance its utility. As you will see in the per member per month cube example, custom rollup can dominate the purpose of a cube making it a challenge to include additional measures without breaking the original calculations.

Suppose that you are dealing with electronic medical record (EMR) data for a set of medical clinics in a 5 county region of your state. You want to determine which physicians and patients are giving and receiving treatments within the recommended time frame and those that are going beyond recommended periods without regular visits. To complicate matters, you know from examining your data that patients are able to visit any of the clinics in the system, visit a variety of doctors, and some doctors work in multiple clinics. This many-to-many relationship of providers and

patients can be a challenge when building a cube. In this scenario, you should break the problem down into separate cubes, complete these cubes, and then attempt to combine the aggregations into a single cube. You should only create the combined cube if you know that your users have need for it. Here is a plan for separate aggregations:

Cube 1: County Clinic Physician Patient
Cube 2: County Clinic Patient
Cube 3: Patient

Since patient visits are spread out over time, you would likely have to pre-summarize the source data to compute a metric that can serve as input to a measure. This process is sometimes called “bucketing” the data. In this case you might assign a value of Q1=1 to a patient if they had a visit on schedule in the first quarter of the year or a value of Q1=0 if they did not. If you did this for 4 quarters in the year then you would have the option to create a measure summing the on-schedule patients in each quarter or average the scores for each patient across the time periods. Plan the pre-summarization carefully depending on which cube you are building. In the case of Cube 1 you want to assign Q1=1 only if the patient visited a particular doctor at a particular clinic. In the case of Cube 3 you can feel free to look across all clinics and physicians.

Sample Data And Test Cube Rollup

Many times you will find that you are working with large amounts of data some of which can be inaccurate or imprecise. It is best to take a small sample of your source data and use only that data for the initial cube development. This will allow you to focus on the cube construction and not be distracted by long processing times and peculiar rollup values in the measures. Once the cube is constructed using the subset of source data, verify the calculations by hand or using SQL or Excel. It is important to test the results because it is easy to get confused by the multiple dimensions and various aggregation hierarchies you are wielding.

Don’t forget that testing is an important step to delivering an accurate cube to end users. In many cases, they will not have good alternative data sources against which to verify the conclusions produced by your cubes. Think about the implications of inaccurate aggregate results. Will the number provided at the top level of the cube lead the CIO or CFO to the wrong conclusion? Does the Chief Actuary depend on this cube report for computing reserves?

Olap Cube Examples

PER MEMBER PER MONTH CUBE

Health insurance companies are required to perform actuarial analysis of membership and claims data related to their products and services. By consolidating claims and membership into a centralized data warehouse, actuarial staff can use SAS BI tools to compute formulas and calculate key metrics. One key metric is the Per Member Per Month (PMPM) calculation.

This calculation is essentially the average monthly cost of insurance claims per person in a plan. Since claims are added to the system daily, the PMPM value changes depending on when you perform the valuation. It has an embedded time component which makes it interesting for actuaries to chart over time. Health plan membership can be viewed in a product hierarchy making it a natural fit for display in an OLAP cube. You can imagine decision makers finding it interesting to drill down and compare Commercial plans with Individual plans and then, within Commercial plans, to compare plan types such as Preferred Provider Organization (PPO) and Health Maintenance Organization (HMO). Within plan types they are interested in comparing Plan A to Plan B.

- **Commercial**
  - PPO
    - Plan A
    - Plan B

- **Individual**
  - HMO
    - Plan C
    - Plan D

An OLAP cube can easily sum up the cost of claims and the membership within each plan. In order to compute PMPM, you have to create a custom measure by dividing the total claims paid in a period by the total number of individuals in the plan known as the membership. Seems simple right? Let's look at the data.

The claim table is presented in Display 1. The first two columns in this table contain date information which is not important for this discussion. Following those are four category columns which form a natural hierarchy: PayorClass, ProductGroup, Product, and Category. These will make up the drill path in the cube. The last column is a total dollar figure of claims paid within that plan, product, and category of service for the year 2009.

VIEWTABLE: Clmlib.Claim

	ValuationDate	Year	PayorClass	ProductGroup	Product	Category	AmountPaid
1	01DEC2010	2009	COMMERCIAL	HMO	Plan H25	Inpatient	$6115672.98
2	01DEC2010	2009	COMMERCIAL	HMO	Plan H25	Outpatient	$2563177.40
3	01DEC2010	2009	COMMERCIAL	HMO	Plan H25	Primary	$808,987.36
4	01DEC2010	2009	COMMERCIAL	PPO	Plan A	Inpatient	$21080968.35
5	01DEC2010	2009	COMMERCIAL	PPO	Plan A	Outpatient	$7758302.86
6	01DEC2010	2009	COMMERCIAL	PPO	Plan A	Primary	$1398248.23
7	01DEC2010	2009	COMMERCIAL	PPO	Plan B	Inpatient	$4238987.36
8	01DEC2010	2009	COMMERCIAL	PPO	Plan B	Primary	$665,012.36
9	01DEC2010	2009	COMMERCIAL	PPO	Plan C	Primary	$125,655.36
10	01DEC2010	2009	INDIVIDUAL	HSA	Plan 804	Primary	$98,987.36
11	01DEC2010	2009	INDIVIDUAL	HSA	Plan 810	Primary	$102,815.36
12	01DEC2010	2009	INDIVIDUAL	PPO	Plan 1020	Primary	$1085547.28

Display 1. Claim Table Listing

The next table is the member table. This table contains the same date and category columns as found in the claims table which will allow you to join the two tables together.

VIEWTABLE: Clmlib.Member

	ValuationDate	Year	PayorClass	ProductGroup	Product	MemberSum
1	01DEC2010	2009	COMMERCIAL	HMO	Plan H25	4520
2	01DEC2010	2009	COMMERCIAL	PPO	Plan A	17313
3	01DEC2010	2009	COMMERCIAL	PPO	Plan B	6208
4	01DEC2010	2009	COMMERCIAL	PPO	Plan C	1021
5	01DEC2010	2009	INDIVIDUAL	HSA	Plan 804	412
6	01DEC2010	2009	INDIVIDUAL	HSA	Plan 810	790
7	01DEC2010	2009	INDIVIDUAL	PPO	Plan 1020	3012

Display 2. Member Table Listing

If you look closely at the member table you will notice that it does not contain all of the category columns from the claim table. The column denoting the category of service is absent. This makes sense because you can be a member in plan without having any claims. It would make little sense to say “there are 425 Outpatient members in Plan A”, for example. Since the Category column is missing, you are forced to omit that from the SQL join that combines these tables.

Here is the code to join claim with member:

```
proc sql;
create table clmlib.detail as select c.*, m.MemberSum as members from
    clmlib.claim as c left join clmlib.member as m on
    c.ValuationDate = m.ValuationDate and
    c.Year = m.Year and
    c.PayorClass = m.PayorClass and
    c.ProductGroup = m.ProductGroup and
    c.Product = m.Product;
quit;
```

This PROC SQL step results in the detail table shown in Display 3.

VIEWTABLE: Clmlib.Detail

	ValuationDate	Year	PayorClass	ProductGroup	Product	Category	AmountPaid	members
1	01DEC2010	2009	COMMERCIAL	HMO	Plan H25	Inpatient	$6115672.98	4520
2	01DEC2010	2009	COMMERCIAL	HMO	Plan H25	Outpatient	$2563177.40	4520
3	01DEC2010	2009	COMMERCIAL	HMO	Plan H25	Primary	$808,987.36	4520
4	01DEC2010	2009	COMMERCIAL	PPO	Plan A	Inpatient	$21080968.35	17313
5	01DEC2010	2009	COMMERCIAL	PPO	Plan A	Outpatient	$7758302.86	17313
6	01DEC2010	2009	COMMERCIAL	PPO	Plan A	Primary	$1698248.23	17313
7	01DEC2010	2009	COMMERCIAL	PPO	Plan B	Inpatient	$4238987.36	6208
8	01DEC2010	2009	COMMERCIAL	PPO	Plan B	Primary	$665,012.36	6208
9	01DEC2010	2009	COMMERCIAL	PPO	Plan C	Primary	$125,655.36	1021
10	01DEC2010	2009	INDIVIDUAL	HSA	Plan 804	Primary	$98,987.36	412
11	01DEC2010	2009	INDIVIDUAL	HSA	Plan 810	Primary	$102,815.36	790
12	01DEC2010	2009	INDIVIDUAL	PPO	Plan 1020	Primary	$1085547.28	3012

Display 3. Detail Table from Claim Join with Member

At this point a challenge arises because claims are recorded at a more granular level than membership. An individual purchases a plan, but an insurance claim is for a specific kind of medical treatment and filed against a given plan. The actuary sees the categorization of the medical treatment as further levels in the drill path, so they need to have qualifiers such as inpatient and outpatient. In practice, PMPM cubes may contain additional levels which require even more fine-grained drill down.

In Display 3, the red circle highlights the duplication of the member count which occurs when you join the more finegrained claim table with the member table. This wrinkle becomes a problem if you attempt to compute the PMPM fraction at the bottom of the hierarchy and then roll up. If you define PMPM as a simple fraction in your PROC OLAP code it will calculate correctly at the lowest level:
COMMERCIAL HMO Plan H25 Inpatient: PMPM = $6,115,672.98 / 4520 = $1,353.02

But it will calculate incorrectly when rolled up because the members will be summed.

COMMERCIAL HMO Plan H25 : PMPM = ($6,115,672.98 + $2,563,177.40 + $808,987.36) / (4520 + 4520 + 4520) = $9,487,837.74 / 13,560 = $699.69

The plan does not have 13,560 members so this is incorrect. The proper calculation is:

COMMERCIAL HMO Plan H25 : PMPM = ($6,115,672.98 + $2,563,177.40 + $808,987.36) / 4520 = $9,487,837.74 / 4520 = $2,099.08

Summing of the members at the lower levels of the hierarchy is wrong. But summing the member at upper levels of the hierarchy is correct. To state the problem in technical terms: We want the cube to roll up the numerator using a regular sum aggregation, but we need it to roll up the denominator with an awareness at lower levels to keep the member sum constant and at upper levels to perform a regular sum aggregation.

In order to accomplish this custom aggregation, you need to aggregate the numerator and denominator portions of the PMPM in a pre-processing step. Then you feed the aggregated data to PROC OLAP in a series of tables - one table for each level in the drill path. Starting with the detail table and using PROC SQL, create an aggregate table for each level, bottom up:

```
proc sql noprint;
    create table clmlib.agg1 as
        select PayorClass, ProductGroup, Product, Category,
            max(members) as members,
            sum(AmountPaid) as paidtotal
        from clmlib.detail
        group by PayorClass, ProductGroup, Product, Category;

   create table clmlib.agg2 as
        select PayorClass, ProductGroup, Product,
            max(members) as members,
            sum(paidtotal) as paidtotal
        from clmlib.agg1
        group by PayorClass, ProductGroup, Product;
```

```
    create table clmlib.agg3 as
        select PayorClass, ProductGroup,
            sum(members) as members,
            sum(paidtotal) as paidtotal
        from clmlib.agg2
        group by PayorClass, ProductGroup;

    create table clmlib.agg4 as
        select PayorClass,
            sum(members) as members,
            sum(paidtotal) as paidtotal
        from clmlib.agg3 group by PayorClass;
quit;
```

The first aggregation table contains all of the category variables. The max function is used to prevent membership from summing. In this case, the agg1 table is the same as the detail table that was passed in, but column names are changed slightly. The second aggregation drops the Category column and uses agg1 as input. When moving to the next aggregation, rolling up further, the max function is swapped for a sum function because this is the level where you need to begin summing up the membership. This is the first time that a membership category column has been rolled up. From this point up to the highest aggregation, the sum function is used to roll up membership. For the total claim payments, a simple sum is used at all levels. You might be tempted here to create all agg levels from the detail table. Don't do that because the sum of members will be incorrect at the top levels. Building each agg table using the level below it ensures that the lower level member counts are combined correctly and membership is not counted more than once.

The next step in this solution is to create PROC OLAP code and customize it to include the PMPM calculation. If you are new to PROC OLAP, it is best to build a simple cube using the agg1 table within SAS OLAP Cube Studio. This tool will guide you through building a cube and will allow you to save the generated PROC OLAP code. Build a simple cube that sums the paidtotal column, test the cube to confirm the drill hierarchy and then copy the code into either base SAS or Enterprise Guide. Using Enterprise Guide is a good idea because it has a user-friendly OLAP Cube viewer. You can easily move between code changes and cube viewing as you iterate on your solution.

Next, you need to remove the instruction from the PROC OLAP statement indicating agg1 as the detail table. Just comment out the data= option:

```
PROC OLAP /*data = clmlib.agg1*/
```

Add in aggregation statements for all levels, telling PROC OLAP the tables that should be used at each level in the drill hierarchy.

```
AGGREGATION PayorClass ProductGroup Product Category / table = clmlib.agg1 ;
AGGREGATION PayorClass ProductGroup Product / table = clmlib.agg2 ;
AGGREGATION PayorClass ProductGroup / table = clmlib.agg3 ;
AGGREGATION PayorClass / table = clmlib.agg4 ;
```

Create measure statements pointing to the pre-aggregated metrics in the agg tables.

```
MEASURE __members column=members stat=sum
                    desc='Member Count' aggr_column=members;

MEASURE __paidtot column=PaidTotal stat=sum
                    desc='Total claims' aggr_column=paidtotal;
```

Finally, use the computed measures as input to calculated members

```
/*These are the three measures that we want to have in our cube*/

DEFINE MEMBER "[PMPM].[measures].[Total Claims Paid]" as
                "([measures].[__paidtot]) , FORMAT_STRING = 'dollar32.2'";

DEFINE MEMBER "[PMPM].[measures].[Member Count]" as
                "([measures].[__members]) , FORMAT_STRING = 'COMMA32.' " ;

DEFINE MEMBER "[PMPM].[measures].[Per Member Per Month]" as
                "([measures].[__paidtot] / [measures].[__members]) ,
                    FORMAT_STRING = 'dollar32.' " ;
```

The last member listed here is the PMPM fraction. Notice that the code instructs the cube to divide __paidtot by __members. Since you have already computed the numerator and denominator correctly, the cube can handle the division. The double underscore prefix has no special meaning. It was added by the programmer to easily distinguish the multiple places where similar column and measure names are used. The resulting cube has three formatted members: Total Claims Paid, Member Count, and PMPM. It may not be necessary to show the total claims and the member count, but they are included here in order to visually inspect the roll up values and perform hand checking of the cube calculations.

Display 4 shows the resulting cube as viewed within Enterprise Guide.

View 1 | View 2

Roles Show Filters Highlights

Columns: Member Count | Total Claims Paid | Per Member Per Month

Rows: Claims > All Claims

				Measures	Member Count	Total Claims Paid	Per Member Per Month
PayorClass	ProductGroup	Product	Category				
COMMERCIAL					29,062	$45,055,012.26	$1,550
	HMO				4,520	$9,487,837.74	$2,099
		Plan H25			4,520	$9,487,837.74	$2,099
	HMO		Inpatient		4,520	$6,115,672.98	$1,353
		Plan H25	Outpatient		4,520	$2,563,177.40	$567
			Primary		4,520	$808,987.36	$179
	PPO				24,542	$35,567,174.52	$1,449
		Plan A			17,313	$30,537,519.44	$1,764
COMMERCIAL			Inpatient		17,313	$21,080,968.35	$1,218
		Plan A	Outpatient		17,313	$7,758,302.86	$448
			Primary		17,313	$1,698,248.23	$98
	PPO	Plan B			6,208	$4,903,999.72	$790
			Inpatient		6,208	$4,238,987.36	$683
		Plan B	Primary		6,208	$665,012.36	$107
		Plan C			1,021	$125,655.36	$123
		Plan C	Primary		1,021	$125,655.36	$123
INDIVIDUAL					4,214	$1,287,350.00	$305
	HSA				1,202	$201,802.72	$168
		Plan 804			412	$98,987.36	$240
	HSA	Plan 804	Primary		412	$98,987.36	$240
		Plan 810			790	$102,815.36	$130
INDIVIDUAL		Plan 810	Primary		790	$102,815.36	$130
	PPO				3,012	$1,085,547.28	$360

Display 4. PMPM Cube in Enterprise Guide

Notice the areas circled in red. You can see that the member count is repeated across the service categories within a plan, and the total plan member count of 24,542 is the correct sum. Now notice the area circled in green. Despite the constant value for membership, the actuary can see a proper, variable PMPM calculation down to the service category level, allowing him/her to compare Inpatient and Outpatient service costs.

Patient Data Cube

This example uses public domain data from a clinical trial for a drug named Nicardipine. The NICSAH studies, as they are called, contain a wide variety of data as well as a time dimension which represents the duration of the study. In this example you will see just a small slice of the data from one of the NICSAH studies. The goal of this example is to build a cube that will present a meaningful analysis of patient vital sign measurements.
The first input data set contains demographic information for the enrolled patients. Per CDISC convention, demographic data is stored in a data set named DM.

VIEWTABLE: Nicsah.Dm

	Study Identifier	Domain Abbreviation	Unique Subject Identifier	Date/Time of Birth	Age	Sex	Race	Description of Planned Arm
1	NICSAH1	DM	101001	1924-03-02	63	F	WHITE	Placebo
2	NICSAH1	DM	101002	1921-08-11	66	M	WHITE	NIC .15
3	NICSAH1	DM	101003	1956-08-03	31	F	BLACK OR AFRICAN AMERICAN	Placebo
4	NICSAH1	DM	101004	1939-08-17	48	F	WHITE	NIC .15
5	NICSAH1	DM	101005	1920-11-14	67	F	WHITE	Placebo
6	NICSAH1	DM	101006	1955-08-10	32	M	BLACK OR AFRICAN AMERICAN	Placebo
7	NICSAH1	DM	101007	1925-05-29	63	M	WHITE	NIC .15
8	NICSAH1	DM	101008	1921-06-26	67	M	WHITE	NIC .15
9	NICSAH1	DM	101009	1950-06-17	38	F	BLACK OR AFRICAN AMERICAN	NIC .15
10	NICSAH1	DM	101010	1939-09-08	48	F	BLACK OR AFRICAN AMERICAN	Placebo
11	NICSAH1	DM	101011	1955-02-03	33	F	BLACK OR AFRICAN AMERICAN	NIC .15
12	NICSAH1	DM	101012	1939-07-03	49	M	WHITE	Placebo
13	NICSAH1	DM	101013	1956-03-17	32	M	WHITE	Placebo
14	NICSAH1	DM	101014	1914-07-23	74	F	WHITE	NIC .15
15	NICSAH1	DM	101015	1942-02-28	47	F	WHITE	Placebo
16	NICSAH1	DM	101016	1911-03-20	78	F	BLACK OR AFRICAN AMERICAN	NIC .15
17	NICSAH1	DM	101017	1953-07-09	35	M	WHITE	NIC .15
18	NICSAH1	DM	11001	1968-10-24	18	M	WHITE	Placebo

Display 5. DM Table Listing

This data set contains one record per patient and each patient can be uniquely identified by the column USUBJID, seen here under the label “Unique Subject Identifier”. The DM data set contains information describing the patient in demographic terms including age, sex and race.

The second data set contains vital sign measurements collected during the trial. Each row in the VS data set denotes a single vital sign measurement.

VIEWTABLE: Nicsah.Vs

	Study Identifier	Domain Abbreviation	Unique Subject Identifier	Vital Signs Test Short Name	Vital Signs Test Name	Numeric Result/Finding in Standard Units	Standard Units	Date/Time of Measurements	Study Day of Vital Signs
1	NICSAH1	VS	101001	SYSBP	Systolic Blood Pressure	142	mmHg	1988-01-21T17:15:00	1
2	NICSAH1	VS	101001	DIABP	Diastolic Blood Pressure	64	mmHg	1988-01-21T17:15:00	1
3	NICSAH1	VS	101001	HR	Heart Rate	86	BEATS/MIN	1988-01-21T17:15:00	1
4	NICSAH1	VS	101001	SYSBP	Systolic Blood Pressure	150	mmHg	1988-01-21T17:30:00	1
5	NICSAH1	VS	101001	DIABP	Diastolic Blood Pressure	70	mmHg	1988-01-21T17:30:00	1
6	NICSAH1	VS	101001	HR	Heart Rate	88	BEATS/MIN	1988-01-21T17:30:00	1
7	NICSAH1	VS	101001	SYSBP	Systolic Blood Pressure	152	mmHg	1988-01-21T17:45:00	1
8	NICSAH1	VS	101001	DIABP	Diastolic Blood Pressure	68	mmHg	1988-01-21T17:45:00	1
9	NICSAH1	VS	101001	HR	Heart Rate	88	BEATS/MIN	1988-01-21T17:45:00	1
10	NICSAH1	VS	101001	SYSBP	Systolic Blood Pressure	154	mmHg	1988-01-21T18:00:00	1
11	NICSAH1	VS	101001	DIABP	Diastolic Blood Pressure	72	mmHg	1988-01-21T18:00:00	1
12	NICSAH1	VS	101001	HR	Heart Rate	90	BEATS/MIN	1988-01-21T18:00:00	1
13	NICSAH1	VS	101001	SYSBP	Systolic Blood Pressure	138	mmHg	1988-01-21T18:15:00	1
14	NICSAH1	VS	101001	DIABP	Diastolic Blood Pressure	64	mmHg	1988-01-21T18:15:00	1
15	NICSAH1	VS	101001	HR	Heart Rate	72	BEATS/MIN	1988-01-21T18:15:00	1
16	NICSAH1	VS	101001	SYSBP	Systolic Blood Pressure	128	mmHg	1988-01-21T19:00:00	1
17	NICSAH1	VS	101001	DIABP	Diastolic Blood Pressure	60	mmHg	1988-01-21T19:00:00	1
18	NICSAH1	VS	101001	HR	Heart Rate	62	BEATS/MIN	1988-01-21T19:00:00	1
19	NICSAH1	VS	101001	SYSBP	Systolic Blood Pressure	138	mmHg	1988-01-21T20:15:00	1
20	NICSAH1	VS	101001	DIABP	Diastolic Blood Pressure	70	mmHg	1988-01-21T20:15:00	1

Display 6. VS Table Listing

Here again, you see the USUBJID column, making it quite easy for you to join the vital sign records to the appropriate patient record in the demographic data table. The column labeled “Numeric Result/Finding in Standard Units” contains the data of interest. The values in this column are the actual vital sign measurements. The column is named VSSTRESN which is shorthand for “vital sign, standard, result, numeric”. The data in this column will form the basis of the metrics in the NICSAH cube.

The demographic data contains a variety of sensible options for cube dimensions. You can segment the trial population into various cohorts using age bands, race, sex. The challenge of the patient data cube comes when you consider how to roll up the vital sign measurements. You cannot add blood pressure or heat rate values. You could average the vital sign measurements, but an average is a blunt instrument and will not provide much insight for clinical reviewers, especially when rolled up into larger patient groupings. One answer is to interpret the vital sign measurement and create a calculated column that can be rolled up. Take the example of blood pressure and ask yourself, “What is important to know about blood pressure?” The answer is that you want to know if the patient’s pressure is high, low, or normal. Does it make sense to analyze this across a population? Yes, it does. Perhaps knowing the percentage of patients within a given population having high, low, and normal blood pressure could be very useful. In the case of the NICSAH study, the clinical reviewer could use this information to compare one cohort to another. If this same technique were applied to data collected from electronic medical records at a hospital or clinic, you could use the results to compare one physician to another or one clinic to another.

The next step in building the cube is to join the VS data to the patient records in the DM table. This is done with a simple left join on USUBJID. After joining the data, make a pass over the resulting table with a data step and

compute three new columns: VSLOW, VSNORMAL, VSHIGH. These columns contain a 1 or 0 depending on the range of the VSSTRESN value. Add another column called PATIENT_COUNT which will form the denominator of the percentage calculation. Here is a portion of the data step code that computes the high, low, normal calculated values for each vital sign measurement.

```
/*Every record gets a patient count of 1*/
patient_count = 1;

/*default range is Normal*/
vshigh = 0;
vsnormal = 1;
vslow = 0;

/*IMPORTANT: These figures are not intended as medical advice.*/
if (vstestcd = 'SYSBP' and vsstresn > 140) then do;
    vshigh = 1; /*High systolic BP*/
        vsnormal = 0;
end;
if (vstestcd = 'SYSBP' and vsstresn < 90) then do;
    vslow = 1; /*Low systolic BP*/
        vsnormal = 0;
end;
else if (vstestcd = 'DIABP' and vsstresn > 90) then do;
vshigh = 1; /*High diastolic BP*/
vsnormal = 0;
end;
else if (vstestcd = 'DIABP' and vsstresn < 60) then do;
    vslow = 1; /*Low diastolic BP*/
        vsnormal = 0;
end;
else if (vstestcd = 'HR' and vsstresn > 160) then do;
    vshigh = 1; /*High heart rate*/
        vsnormal = 0;
end;
else if (vstestcd = 'HR' and vsstresn < 50) then do;
    vslow = 1; /*Low heart rate*/
        vsnormal = 0;
end;
```

Once you have created calculated columns with 1's and 0's, you can proceed to create a cube that will add them up and compute the percentage of patients in each range. Create the cube using a typical PROC OLAP invocation and include these measure and member statements:

```
MEASURE __vslow column=vslow stat=sum desc='Low range';
MEASURE __vsnormal column=vsnormal stat=sum desc='Normal range';
MEASURE __vshigh column=vshigh stat=sum desc='High range';
MEASURE __patcount column=patient_count stat=sum desc='Total claims';

/*Calculate percent high, percent low, percent normal*/
DEFINE MEMBER "[VITALS].[measures].[Percent patients high]" as
                "([measures].[__vshigh] / [measures].[__patcount]) ,
                FORMAT_STRING = 'percent6.1'";

DEFINE MEMBER "[VITALS].[measures].[Percent patients normal]" as
                "([measures].[__vsnormal] / [measures].[__patcount]) ,
                FORMAT_STRING = 'percent6.1' " ;

DEFINE MEMBER "[VITALS].[measures].[Percent patients low]" as
                "([measures].[__vslow] / [measures].[__patcount]) ,
                ORMAT_STRING = 'percent6.1' " ;
```

The resulting cube displays a grid of percentages and even includes a time dimension allowing the clinical reviewer to see how the percentages change over the life of the study. Using the features of Enterprise Guide cube viewer, you can drill down into the cube and isolate a specific section of data for closer scrutiny.

View 1 | View 2

Roles Show Highlights

Columns: Patients > All Patients > WHITE | Percent patients high | Percent patients low | Percent patients normal

Rows: Visits > All Visits

Filters:

Sex	F			M		
Measures	Percent patients high	Percent patients low	Percent patients normal	Percent patients high	Percent patients low	Percent patients normal
Visit						
09FEB1988	20%	1.4%	79%	28%	0.0%	72%
10FEB1988	28%	5.1%	67%	8.3%	0.0%	92%
11FEB1988	27%	3.2%	69%	22%	0.0%	78%
12FEB1988	32%	7.9%	60%	29%	0.0%	71%
13FEB1988	30%	4.7%	65%	8.3%	0.0%	92%

Display 7. NICSAH Vital Signs Cube

Behind the scenes, the cube contains several years of data for patients from multiple races, but here we see just a subset of the available data. Display 7 shows white patients during a 5 day segment of the trial and enables the clinical reviewer to compare male and female patients during that period. Drilling down into Male or Female subdivides the patients into age bands.

Conclusion

Business Intelligence for Healthcare is certainly a challenge. Despite the rapid adoption of these technologies by departments focused on routine accounting and business functions, healthcare businesses struggle to create accurate, useful cross-population reports. As this paper has demonstrated, it may require going beyond the stock point-and-click feature set of the BI software in order to produce effective tools for decision makers. SAS Buisiness Intelligence and its OLAP Cube technology have the flexibility to create complex cube-based reports using healthcare data. With a little bit of technical savvy, you can create cubes which provide insight into healthcare cost and quality, enabling a more effective and beneficial enterprise.

References

SAS Global Forum 2010 Proceedings, Seeno,Heather and Leach, Alan , “The Quest for the Holy Grail: OLAP Approaches for Calculating Actuarial Measures”, Paper 162-210, http://support.sas.com/resources/papers/proceedings10/162-2010.pdf

eHow, Bagley, Soren, “How to Calculate PMPM”, http://www.ehow.com/how_5305766_calculate-pmpm.html

Actuary.com, Linfield, Jed, “Calculation of Medical Liability - Combination of Statistical Methods, Actuarial Judgment, and Communication”, Southeastern Actuaries Conference, November 19, 2008, http://www.actuary.com/seac/handouts/medical_liability_fall_2008.pdf