Making Health Data Elegant

Save to myBoK

By Matthew Moschel, CHDA

Too often, data analysts make life difficult for themselves. An analyst may produce code that is much longer and more complicated than it needs to be, causing a coworker to re-run code for several hours, not including the execute time. Why? These challenges stem from three root sources of coding frustration:

1. Failure to keep relational data relational
2. Failure to plan appropriately for an analysis
3. Unnecessary replication of code

Before continuing on how to fix this, some housekeeping. This article requests—but does not require—a good understanding of relational databases, a familiarity with health data structures, and a general understanding of code manipulation in SQL, SAS, or Access. Examples will be presented in T-SQL, and the advice below could relate to logic running on a production server or an ad hoc report. Transact-SQL is Microsoft's extension to SQL (structured query language) and is the syntax used to query, alter, and define databases within Microsoft SQL Server.

Keeping Data Flexible

Quality control (QC) often involves checking the work of coworkers to ensure the accuracy of data analytics. In the healthcare litigation consulting field, QC is a part of daily life. It is expected that scripts of logic will be produced to the opposing legal side, and thus all code must be (a) without error, (b) well organized, and (c) thoroughly QC'ed.

Take the following hypothetical scenario: A coworker has written a year-over-year damages calculation in which she began with relational data, used a sum (case when year = 201_) to place data into 12 yearly columns, and then continued to work with that non-relational data for 2,000 lines of code. Each select clause references all 12 columns, and every subsequent calculation references all 12 years. Given the volume and duplicity of code, replicating the logic would be the only method to confirm the absence of typos or inadvertent exclusions. Based on experience, the team will have to run this code again, as the ever-progressing variable of time inevitably decides 2016 is now a thing. Does anyone have the patience to comb through this script and add a new year to every step?

The point—keep data relational until the final point of extraction. QC professionals will be thankful, and zero lines of code will need to be changed when the clock rolls over at midnight on December 31. How is this possible while writing such a complicated script? Allow the code to correctly handle all years by joining the year to year + 1 any time the calculation requires a year-over-year comparison.

Figure 1

Example in which claims with more than one reason could be double counted.

***Claim-Line* Reasons**	**No. of *Claims***
Reason 1	3

Reason 2	5
Reason 3	2
Total	6

Planning for the Analysis

Admittedly, "planning for the analysis" sounds 100 percent cliché. What analyst doesn't plan for his or her analysis? But why then does it sometimes feel like duct tape is necessary to squeeze data through a non-logical script? Three questions should remain in the back of an analyst's mind before writing a single line of code—what is the intended output; is the data understood; and does the data "flow"?

What is the Intended Output?

Does the analyst understand exactly what the output will look like? Has he or she sketched the output and thought about the nuances? Is he or she attempting to count distinct (i.e., unique) claim numbers while grouping by the claim-line descriptions? Will he or she understand that a claim with more than one distinct claim-line description will be counted in separate buckets, and thus the sum of the parts will be greater than the total number of claims in the universe?

Unfortunately, this section contains more questions than answers. The analyst will need to fully evaluate the data prior to running analytics and assess whether the intended output makes sense. For simplistic analysis of familiar data, it may be possible to identify these nuances before sitting down at the keyboard. Other times, intricacies may appear that require the involvement of a manager, client, or both. Depending on the results of that involvement, analysis may require assumptions, documentation, and a caveat upon viewing the output.

Is the Data Understood?

The relative skill of an analyst is not determined based on his or her ability to write syntax; any analyst could add additional logic by making a script longer. Data skills are based on that person's understanding and ability to accurately interpret and manipulate data. The best manipulations are those that make a script more functional while using less complexity, resulting in a more elegant solution for the data at hand.

To be competent with SQL (or any coding language) means being skilled at interpreting previously unfamiliar data, and includes the ability to give valuable insights to that data without prior training. To the extreme, an analyst who does not understand the data may produce erroneous reports by simply failing to test the primary key of a "join."

These abilities come with experience but specific questions must be asked of nearly every dataset:

- Are the primary keys understood?
- Could the data be segmented and therefore made more relational? (Hint: It should probably be made more relational.)
- Is the analysis following a direct route to the desired answer?

Knowing the answers to these questions can make coding much more efficient. If primary keys are known, an analyst need not worry about record duplication or ambiguous return values on a select or update statement while executing a join.

Does the Data "Flow"?

While most logic would flow seamlessly from one step to the next, certain occasions might tempt the alteration of data outside this logical flow. For example, the analysis may result in a table of records to exclude in step 3 of Figure 2, but the actual

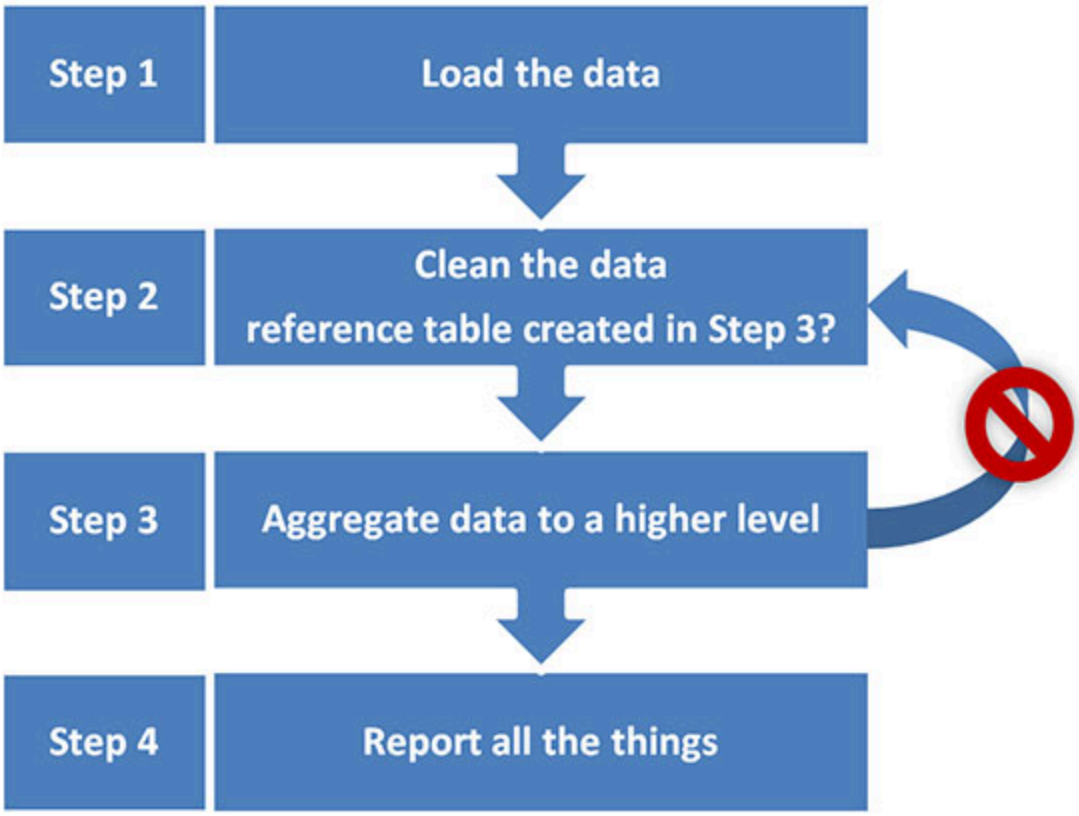
exclusion needs to occur in a "where clause" of step 2. This might be the fastest route to the answer, but executing this process again would require segmented selection of code and running specific lines of code more than once.

This process of circular execution of scripts should be avoided at all times. When you pare down the analytics at hand, it may become apparent that step 3 can be moved closer to the start to avoid this circular logic. Alternatively, if steps 2 and 3 need to run twice, these items could be written to a stored procedure with alternate inputs allowing variable data to be processed. Maintaining the logical progression of analytics will allow the team to get up to speed quickly when new data becomes available.

Further, a coherent sequence may prevent others from questioning the otherwise curiously confusing logic—particularly in the courtroom. Circular styles of writing code should be strictly prohibited, but many analysts continue to offend by committing this faux pas.

Figure 2: Step-by-Step Analytics

From the most simplistic summary report to the most complex calculation, data analytics at its heart is a logical step-by-step process.



Code Once Use Many

Certain types of health data analytics (i.e., RAPS, MMR, MOR, 837) involve multiple extractions and productions of data. In these cases, it can be tempting to take a newly received dataset and run it through the same logic after employing a find/replace function to reference the new data table. This can result in the same script being copied repeatedly to alter data that is otherwise in an identical format. Problems arise when changes to the logic are required and need to be made and executed in all copied scripts, despite the fact that they perform identical actions.

Best practice is to employ a "Code Once" internal policy allowing identically structured data to be processed using just one script and table. In some scenarios, this methodology requires the use of stored procedures or functions to execute on various tables. Other times, similar data is unioned, and a "source" column is added to track initial locations of received data. Both

methods reduce the volume of code, and the result is a more elegant and streamlined analytics process that is more flexible to unsolidified requirements.

Simple Can Be Better with Analytics

An outside observer, upon viewing these scripts for the first time, might think, “This logic looks utterly simplistic! Variables referenced only once!? Data flowing through in such a streamlined manner!?” This same observer might also conclude that the underlying logic is elementary.

Deeper inspections of the code might reveal the truth—the code looks elegant because it was designed to be elegant from the start. A complex damages calculation might occur as a simple set of insert statements, declaring variable proportions of previously iterated summations from an intentionally changing view.

The three ideas—keeping relational data relational, planning for your analysis, and the “Code Once” initiative—can make complex analytics feel as though SQL or SAS was designed specifically for the data at hand. The complex and duct-taped script can be relegated to the past, with logic purposefully written to perform the specific required actions for the output. The flexible and elegant script can be showcased, especially in court, as the reason for such confident conclusions.

Matthew Moschel (mmoschel@thinkbrg.com) is a consultant in the Tampa, FL, office of Berkeley Research Group. The views and opinions expressed in this article are those of the author and do not necessarily reflect the opinions, position, or policy of Berkeley Research Group, LLC or its other employees and affiliates.

Article citation:
Moschel, Matthew. "Making Health Data Elegant" *Journal of AHIMA* 86, no.10 (October 2015): 32-35.