PROC SQL: A Powerful Tool to Improve Your Data Quality
Keh-Dong Shiang, City of Hope National Medical Center, Duarte, CA

ABSTRACT
The Structured Query Language (SQL) is a standardized query language that can be efficiently used to retrieve and update data stored in relational databases. However, SQL’s WHERE-clause row-selection feature can be interestingly utilized for data validation and error checks. In this paper, I will present a method of designing a highly structured validation program, which is applied to implement a data quality assurance procedure for collecting clean clinical data in the Diabetes department at City of Hope National Medical Center. With both SAS® macros and SQL procedures used, the results will demonstrate the following benefits: easy database maintenance with no hard-coded checks, transparency of sophisticated error-check edits to the users, effectively-geared with nested structures, easy-to-understand reporting system, and considerable reduction of the total programming time required.

INTRODUCTION
Data validation is the examination of data to determine conformance and consistency according to the needs of the data users. It is also an essential quality assurance process to ensure that data collected and subsequently used in making management decisions are free of errors, and that inappropriate information which could lead to misinterpretation or erroneous conclusions is eliminated. The primary focuses of data validation are accuracy, completeness, precision, reliability and integrity of individual data values so that the collected values can be trusted and further analyzed. In general, data validation should be an ongoing process in the lifespan of any data-gathering program.

The method presented in this paper mostly involves the identification and presentation of faulty records by utilizing PROC SQL tool in a sample database. As a result, this validation method can establish documented evidence providing a high degree of assurance that a specific computerized process will consistently produce good quality resultant data sets meeting your predetermined specifications.

WHAT ARE SQL AND WHERE CLAUSE?

SQL
SQL permits users to access data in relational database management systems, such as SAS, Oracle, Informix, Sybase, Microsoft Access, SQL Server, and others, by allowing users to describe the data they wish to see as well as to define the data and manipulate it within the database.

THE WHERE CLAUSE
The WHERE clause, added to the SELECT statement, is used to specify that only certain rows of the table are displayed, based on the criteria described in that WHERE clause.

DIFFERENCE, EQUIVALENCE AND TERMINOLOGY
To clearly identify the differences (i.e. equivalencies) among PROC SQL, DATA step and common data file processing, the following components among terms are listed:

<table>
<thead>
<tr>
<th>Data File Processing</th>
<th>SQL</th>
<th>SAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td>Table</td>
<td>Data Set</td>
</tr>
<tr>
<td>Record</td>
<td>Row</td>
<td>Observation</td>
</tr>
<tr>
<td>Field</td>
<td>Column</td>
<td>Variable</td>
</tr>
</tbody>
</table>
CHARACTERISTIC OF A SQL TABLE

- A table is perceived as a two-dimensional structure composed of rows and columns, which is an alias of a SAS data set in SQL.
- Each table row (i.e. tuple) represents a single entity occurrence within the entity set.
- Each table column represents an attribute and has a distinct name.
- Each row/column intersection represents a single data value.

WHY VALIDATION? DATA PROBLEMS?

Here is an example of a problematic data set in the database:

<table>
<thead>
<tr>
<th>FirstName</th>
<th>Gender</th>
<th>Height</th>
<th>DOB</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>F</td>
<td>158</td>
<td>12/34/1967</td>
<td>38</td>
</tr>
<tr>
<td>John</td>
<td>M</td>
<td>175</td>
<td>10/29/1975</td>
<td>30</td>
</tr>
<tr>
<td>Steve</td>
<td>F</td>
<td>3.6</td>
<td>7/12/1983</td>
<td>22</td>
</tr>
<tr>
<td>Nicole</td>
<td>F</td>
<td>164</td>
<td>12/8/1974</td>
<td>27</td>
</tr>
<tr>
<td>Ingrid</td>
<td>F</td>
<td>167</td>
<td>8/12/1974</td>
<td>32</td>
</tr>
</tbody>
</table>

In this table, five faulty records are presented: an incorrect input of “DOB” in row 1 and 2, the wrong “Height” unit applied in row 3, and incorrect FirstName value in the last row. Some of the required values like Gender in row 3 and DOB in row 4 are left blank. The age column is a calculated field, which means the values would be dynamically changed with respect to their corresponding DOB values. However, inconsistencies between DOB and Age in row 2 and row 4 are easily found.

These incorrect and faulty data should be identified and corrected or even removed from the patient record, so that they do not affect the investigator’s decision making while analyzing and reporting the resultant data. Decision-support system that operates on incorrect and faulty data will create incorrect statistical results and generate faulty advice. Therefore, the implementation of a good validation system in the database is essential to ensure that the data records are clean, accurate and complete.

DATA VALIDATION IN MICROSOFT ACCESS

In the design mode of Microsoft Access, you can predefine the ValidationRule property to specify requirements for data entered into a record, field or form control. When data is entered that violates the ValidationRule setting, the ValidationText property can be applied to specify the pop-up message to be displayed to the database users. For more advanced validating techniques, the validation expression can also contain system built-in and user-defined functions as part of field and record validation rules.

USING VALIDATION RULES TO RESTRICT DATA

A validation rule is an input expression that can precisely define the information that will be accepted in a single or multiple fields in a record. You might use a validation rule on a field containing the date a patient was admitted to hospital to prevent a date in the future from being entered. Moreover, if you restrict your patient records to only certain local areas, you could use a validation rule on the phone field or ZIP code field to refuse entries from other areas.

General speaking, validation rules are to help prevent users’ input errors by limiting the data that fields accept. For example, if a clinical protocol study was initiated in 2004, then a validation rule could prevent potential patients having recruitment date recorded before 2004. Another example can occur when you are entering patient follow-up information, which requires an existing patient identifier in the database (i.e. in patient’s profile). If the patient id does not exist, then the database will not allow you to save the patient follow-up information in the patient master table.

DATA VALIDATION IN MICROSOFT EXCEL
Data validation in Excel is a user-friendly tool that helps you control the kind of information that is entered in your worksheet. By choosing "Validation" from the "Data" menu, a Data Validation window with three tabs (i.e. Settings, Input Message and Error Alert) will appear. With this data validation, you can:

- provide a list of choices
- restrict data entries to a specific type or size
- create custom settings

This is a very useful tool and can be easily applied in any popular Excel data system.

EXPANDING MICROSOFT’S VALIDATION CONCEPT TO SAS

The same above idea and concepts can be applied to our clinical data validation case as well as in the SAS environment. In Microsoft office applications, an appropriate ValidationRule, Message and Error Alert setting will show users predefined error messages when invalid data is entered. In both Excel and Access environments, the settings of validation criteria in design mode are allowing or limiting Whole Number, Decimal, List, Date, Time and Text Length. Finally, those concepts have been transformed into an efficient way and expanded in our SAS clinical database system.

VALIDATION TYPES

A list of validation types is shown below:

<table>
<thead>
<tr>
<th>Type of Validation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range Check</td>
<td>This checks that the data lies within a specified range of values</td>
</tr>
<tr>
<td>Presence Check</td>
<td>This checks that the required data is not missing</td>
</tr>
<tr>
<td>Domain Check</td>
<td>This checks that only certain values are accepted</td>
</tr>
<tr>
<td>Cross-Field Check</td>
<td>This checks that multiple fields in combination are valid</td>
</tr>
<tr>
<td>Cross-Table Check</td>
<td>This checks that multiple tables in combination are valid</td>
</tr>
<tr>
<td>Uniqueness Validation</td>
<td>Ensure the values in a column are unique</td>
</tr>
<tr>
<td>Reference Integrity Validation</td>
<td>Validate values between tables in relational database model</td>
</tr>
<tr>
<td>Duplicate Identification</td>
<td>Identify a row as an unwanted duplicate record</td>
</tr>
<tr>
<td>Format Consolidation</td>
<td>Control data values inside a preset mask pattern</td>
</tr>
<tr>
<td>Business Rule Compliance</td>
<td>These are business logics that are specific to the business and unrelated to relational database theory</td>
</tr>
</tbody>
</table>

As you may know, the way two or more tables are joined is to specify the table names in a WHERE clause. Thus, cross-table check can be performed via relatively complex WHERE clause with an appropriate specified restriction. To accomplish this, a subquery, a query-expression that is nested as part of another query-expression, can be used to cross audit the fields between tables. As you know, a subquery will select one or more rows from a table based on values in another table. For instance, the following SAS SQL procedure will find patients who have the unmatched DOB values.

PROC SQL;
SELECT TrackingNo, FirstName, DATEPART(DOB) AS DOB FORMAT=MMDDYY10.
FROM Mylib.S01
WHERE DOB ^= (SELECT DOB FROM Mylib.S03
WHERE S01.TrackingNo=S03.TrackingNo)
QUIT;

<table>
<thead>
<tr>
<th>TrackingNo</th>
<th>FirstName</th>
<th>DOB</th>
</tr>
</thead>
<tbody>
<tr>
<td>137</td>
<td>SUSAN</td>
<td>09/24/1958</td>
</tr>
</tbody>
</table>
WHAT CAN IT DO FOR YOU?

With the proper validation system implemented, you can:

- Ensure the raw data is accurately entered into a computer readable data set.
- Check that character variables contain only valid values, such as 'M' and 'F' for Gender field.
- Check that numeric variables are within predefined ranges. This will ensure that data outside of the permitted range is not allowed; for example an age of 150 and the systolic blood pressure is not between 80 and 200.
- Check for missing values for variables where complete data is necessary.
- Review and eliminate duplicate data entries. For instance, two records with the same patient identifier and the date of hospital visit might constitute an error.
- Overlook the uniqueness of certain values, such as SSN or patient ID number.
- Define logic checks and cross-field checks. For instance, if the question "Is the patient currently taking anti-hyperglycemic medications?" was answered with 'Yes', then the following fields of agent, dose, units and frequency for all medications used should not be left blank, and vice versa.
- Verify that more complex multi-form rules, such as cross-table check or consistency check, have been followed. For example, if a serious adverse event (SAE) with an assigned toxicity code occurs in one data set (e.g. toxicity form), an observation with the same toxicity code will be expected in another data set (e.g. SAE form). In addition, the date of this event observation must be after the start date and before the end date of the study trial or follow-up time period. Furthermore, it is definite that a patient having a hysterectomy or an ovarian cancer could not be recorded as ‘M’ or ‘male’.
- It will reduce the amount of coding you actually have to write to accomplish complicated data auditing tasks as well as provide you with an easier maintenance method.

EXAMPLES OF SQL WHERE CLAUSES

The WHERE clause in SAS SQL procedure supports many standard comparison, special and logical operators, such as EQ, NE, GT, LT, BETWEEN … AND, IN, IS NULL, LIKE, CONTAINS, EXISTS, AND, OR, NOT … etc.

Here is the table listing some examples I have used:

<table>
<thead>
<tr>
<th>Rule – SQL WHERE Clause</th>
<th>What It Indicates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is Null</td>
<td>Should not be blank</td>
</tr>
<tr>
<td>Is Not Null AND PriIns Not BETWEEN 1 AND 6</td>
<td>PrimaryInsurance must be between 1 and 6</td>
</tr>
<tr>
<td>Is Not Null AND HtInch Is Null</td>
<td>HtFeet indicated, but HtInch was not given</td>
</tr>
<tr>
<td>Is Null AND HtInch Is Not Null</td>
<td>HtFeet was blank, but HtInch should not exist</td>
</tr>
<tr>
<td>Is Not Null AND INDEX(Email,’@’)=0 =1 AND N(PriIns,SupIns)=0</td>
<td>Email does not contain @ character</td>
</tr>
<tr>
<td>=0 AND N(PriIns,SupIns)&gt;0</td>
<td>InsuranceYN was answered Yes, but missing</td>
</tr>
<tr>
<td>Is Not Null AND Gender Not IN(‘M’,’F’)</td>
<td>PrimaryInsurance or SupplementInsurance</td>
</tr>
<tr>
<td>Is Not Null AND Cholesterol &gt; 300</td>
<td>InsuranceYN was answered No, but at least one</td>
</tr>
<tr>
<td></td>
<td>PrimaryInsurance or SupplementInsurance filled</td>
</tr>
<tr>
<td></td>
<td>Gender existed but was Not in (M,F)</td>
</tr>
<tr>
<td></td>
<td>Cholesterol value should be less than 300</td>
</tr>
</tbody>
</table>

BRING THE POWER OF PROC SQL INTO DATA VALIDATION

The type of validation on a single field is often referred to as a single-field check. Just because you have validated all of the individual fields in your table doesn't mean that you have completed validating the table. You know by this point in your validation that what has been entered into a specific field meets the validation rules that you have applied to that field. What you don't know is whether the combination of what has been entered into different fields will make sense to you or not. In order to finish validating your table, cross validate the fields to compare the values entered into two or more fields in your table to see if their contents can be considered valid in combination. Evaluating the values from more than one field is referred to as cross-field checks, and it is usually more difficult to program no matter which database system you use. However, this cross-field check will enforce your business rules that limit when you can only add, modify or delete data while the data is matching and correlating with the values in other related fields.
The expression for a field or record validation rule can refer to not only fields, operators, SAS built-in functions, but also can be extended to customized functions. There is a definite need for a more complex check, while validation becomes more sophisticated, which must be geared at further specified user-defined functions. Here are six SAS built-in functions I have frequently used and some customized functions I have developed:

**SAS BUILT-IN FUNCTIONS**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N()</td>
<td>determines the number of non-missing values in a list of numeric fields</td>
</tr>
<tr>
<td>NMISS()</td>
<td>determines the number of missing values in a list of numeric fields</td>
</tr>
<tr>
<td>MIN()</td>
<td>determines the smallest non-missing value in a list of numeric values</td>
</tr>
<tr>
<td>MAX()</td>
<td>determines the largest non-missing value in a list of numeric values</td>
</tr>
<tr>
<td>RANGE()</td>
<td>computes the range (i.e. highest–lowest) in a list of numeric values</td>
</tr>
<tr>
<td>INDEX()</td>
<td>locates the starting position of a substring in a string</td>
</tr>
</tbody>
</table>

In SAS version 9, there are five new 'ANY' functions: ANYALNUM, ANYALPHA, ANYDIGIT, ANYPUNCT and ANYSPACE. These functions will return the location of the first alphanumeric, letter, digit, punctuation or space in a character string. There are also four new 'NOT' functions: NOTALNUM, NOTALPHA, NOTDIGIT and NOTUPPER. These NOT functions are very similar to the ANY functions except that these functions will return the position of the first character value that is NOT a particular value (alphanumeric, character, digit or uppercase character).

**SAMPLE USER-DEFINED FUNCTIONS**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CountFields()</td>
<td>counts the number of non-null fields</td>
</tr>
<tr>
<td>RangeCheck()</td>
<td>returns True for within range and False for out of range</td>
</tr>
<tr>
<td>Maximum()</td>
<td>returns the maximum of non-null value in a list of fields</td>
</tr>
<tr>
<td>Minimum()</td>
<td>returns the minimum of non-null value in a list of fields</td>
</tr>
<tr>
<td>RangeCheck()</td>
<td>returns a Boolean value to check whether a value is within a range (max~min)</td>
</tr>
<tr>
<td>InValue()</td>
<td>returns a Boolean value to check whether a value is in several discrete values</td>
</tr>
<tr>
<td>IsNumeric()</td>
<td>returns True for field value is numeric</td>
</tr>
<tr>
<td>IsAlpha()</td>
<td>returns True for character string is a English letter</td>
</tr>
</tbody>
</table>

CrossFieldCheck(Table1, Field1, Field2, … )
CrossTableCheck(Table1, Field1, Table2, Field2, ….)

For instance, a sample user-defined function called IsAlpha() can be used to validate a text string to make sure that every character in the string is a letter in the English alphabet.

**DESIGN OF DATA VALIDATION**

We have developed 2 validation programs: one was coded in the SAS environment (see APPENDIX), and the other was coded with Microsoft Access Visual Basic for Application (VBA). These two programs did have the same code schema and flow, but they were written as entirely different language syntax. In this 'Data Validation' section, the same programming schema is simply illustrated below:

1. **CREATE A MASTER TABLE**

   First, create a table called "Master" containing all the study SQL tables (such as PhysicianReport form, VitalSigns form, …) that you want to include in the validation checks. The following is what the table looks like:

<table>
<thead>
<tr>
<th>MainTableName</th>
<th>SQLTableNames</th>
<th>STableNames</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbo_vCrossmatch</td>
<td>dbo_vCrossmatch</td>
<td>svCrossmatch</td>
</tr>
<tr>
<td>dbo_vEligibilityChkList</td>
<td>dbo_vEligibilityChkList</td>
<td>svEligibilityChkList</td>
</tr>
<tr>
<td>dbo_vFollowUp</td>
<td>dbo_vFollowUp</td>
<td>svFollowUp</td>
</tr>
<tr>
<td>dbo_vPhysicalExam</td>
<td>dbo_vPhysicalExam</td>
<td>svPhysicalExam</td>
</tr>
<tr>
<td>dbo_vPhysicianReportMaster</td>
<td>dbo_vPhysicianReportIllness</td>
<td>svPhysicianReportIllness</td>
</tr>
<tr>
<td>dbo_vPhysicianReportMaster</td>
<td>dbo_vPhysicianReportMeds</td>
<td>svPhysicianReportMeds</td>
</tr>
</tbody>
</table>
2. ADD VALIDATION RULES

Create programming specifications that take the requirements (e.g., patient’s age at the time of the transplant = age difference between TransplantDate and BirthDate) and translate them into programming and database terminology (i.e., \( \text{AgeAtTransplant} = \text{INT}((\text{TransplantDate}-\text{BirthDate})/365.25) \)). The calculated Age field is derived from the DOB (Date of Birth) field. Similarly, this English-Computer language translation can be applied to AgeAtDiagnosis, AgeAtOnstudy and all other fields on study forms.

After completing the specification table, these requirements should be exposed to and reviewed by the users (in our case, they are study protocol coordinators and clinical research associates - CRAs), to verify whether the requirements are interpreted correctly or not. This process can limit or avoid the “that is NOT what I need” misinterpretation syndrome, where the users see the ‘misinterpreted’ final output that does NOT quite address their interest and needs.

Corresponding to each row in the above Master table, a S-table with ‘SQL Where Condition’ and ‘Error Message’ is also created by entering the ‘logical or conditional row’ based on the criteria defined by the study coordinator and protocol stated rules. The sample S-table may look like this:

<table>
<thead>
<tr>
<th>TableName</th>
<th>FieldName</th>
<th>WhereCond</th>
<th>ErrorMsg</th>
</tr>
</thead>
<tbody>
<tr>
<td>InitialInfo</td>
<td>TypeDiabYN</td>
<td>&lt;&gt; 1</td>
<td>Question type 1 diabetes was answered “No”</td>
</tr>
<tr>
<td>InitialInfo</td>
<td>DOB</td>
<td>Is Null</td>
<td>DOB field should not be blank</td>
</tr>
<tr>
<td>InitialInfo</td>
<td>HtFt</td>
<td>&lt;&gt; Null AND HtFt Not Between 3 and 7</td>
<td>HeightFeet value should be 3~7</td>
</tr>
<tr>
<td>InitialInfo</td>
<td>HomePhone</td>
<td>&lt;&gt; Null AND LEN(HomePhone) &lt;&gt; 12</td>
<td>Incorrect HomePhone value length</td>
</tr>
<tr>
<td>InitialInfo</td>
<td>Gender</td>
<td>&lt;&gt; Null AND Gender Not IN(‘M’, ’F’)</td>
<td>Gender value was not ‘M’ or ‘F’</td>
</tr>
<tr>
<td>PhysicianReport</td>
<td>Cpeptide</td>
<td>&lt;&gt; Null AND CPeptide &gt; 0.2</td>
<td>Fasting C-Peptide must be less than or equal to 0.2</td>
</tr>
</tbody>
</table>

3. PROGRAM LOGIC

Most importantly, the key concept I would like to present is illustrated in this subsection. There are total three nested iterative loops in the program, which are shown as follows.

```
DO (looping through each row in Master table)
  Code Statements
  Choose a SQL table name from Master table

  DO (looping through each row in S-table)
    Code Statements
    Select a row (by row) with SQL statement from S-table

    DO (looping through each row in target table)
      TESTS SQL table one record at a time
      Execute PROC SQL with WHERE Condition in S-table
      If returns 1, inserting 'errors' into Errors table.
      If returns 0, it has passed the validation check.

    END DO

  END DO

END DO
```

In the innermost loop, the SQL statement is composed of three fields (i.e., in square brackets) from the S-table, which should look like:

```
SELECT COUNT(*) FROM [TableName] WHERE [FieldName] [WhereCond]
```

This standard SELECT SQL statement will be executed one row at a time in the innermost loop.
4. RUN CODE AND OUTPUT RESULTS

When SQL query returns value 1, it means that the tested row fails to pass the validation check. Thus, an 'error' record will be appended into the Errors table. An example of this output table is shown below:

<table>
<thead>
<tr>
<th>ID</th>
<th>TableName</th>
<th>FieldName</th>
<th>ErrorMsg</th>
</tr>
</thead>
<tbody>
<tr>
<td>28530</td>
<td>InitialInfo</td>
<td>HomePhone</td>
<td>Incorrect HomePhone value length</td>
</tr>
<tr>
<td>28530</td>
<td>InitialInfo</td>
<td>E_Mail</td>
<td>E-mail address should include '@' character</td>
</tr>
<tr>
<td>28530</td>
<td>InitialInfo</td>
<td>SSN</td>
<td>SSN value's length is incorrect</td>
</tr>
<tr>
<td>1061</td>
<td>InitialInfo</td>
<td>HtFt</td>
<td>HeightFeet value was left blank</td>
</tr>
<tr>
<td>92819</td>
<td>PhysicianReport</td>
<td>Cpeptide</td>
<td>Fasting C-Peptide must be less than or equal to 0.2</td>
</tr>
<tr>
<td>9053</td>
<td>PhysicianReport</td>
<td>Albumin</td>
<td>Albumin value should be greater than 3.5 g/dL</td>
</tr>
</tbody>
</table>
| 62683 | InitialInfo| TypelDiabYN| Question type 1 diabetes was answered "No"

5. ERROR TRAPPING

It is important to ensure that all the basic and complex checks as well as predefined business rules are successfully passed before the error record insertion is triggered and executed. Failure of any one of the rules will be displayed on an error report.

Invalid data entries and failure to pass the validation check are rejected with clearly predefined and user-friendly error messages (as stated at the last column in the above table) and also trigger a subroutine to set a 'Flag' variable’s value in the SQL table to a negative number, such as -1. If data rows pass the validation check, the FLAG value will be set to a positive number, such as 1. Thus, you will see both the pass and fail record identifiers for all the rows in all the tested tables. For further analytical use, you can easily extract the good records from the tested table by selecting only the rows with FLAG=1.

6. DISTRIBUTING AND REVIEWING ERROR REPORTS

The responsible database administrator should check the correctness of computed results before sending out the error report. An easily understood and clearly organized error report can be routed back to the persons who collected and/or input the data. They may refer to the original form information to validate the computerized data.

PROGRAM DEVELOPMENT LIFE CYCLE (PDLC)

The development of this validation program can be broken down into six steps:

1. Gather program requirements via interaction between users and programmer
2. Create programming criteria specifications and validate the SQL query statements
3. Program development and testing
4. Executing the validation programs and printing error reports
5. User review with acceptance or rejection comments
6. Program integration testing and reporting

Therefore, all these validation processes can be summarized with a simple formula:

Validation = Specification + Programming + Testing + Reporting + Reviewing + Documentation

BENEFITS

The benefits of this validation program:
• Programs are in a highly structured and systematic manner.
• Cost effective – it provides a proper architecture to carry out complex data validation tasks and accelerates completion of data auditing while reducing overall cost.
• Compared with the conventional procedure programming, this validation program can reduce the total amount of programming time needed. For instance, if users want to add further criteria to validate data, the database administrator needs to add ‘logical or conditional rows’ in the S-table. The validation program can be left intact, and the only thing we have to do is to fill in all the specific requirements in the S-table.
• Easy database maintenance, just updating the S-table (creating SQL specifications that take the requirements from users and translate them into database terminology and SQL syntax) to maintain the high quality data.
• Automate data-reporting routine tasks, generating periodic error reports for users to review the faulty records.
• ‘FLAG’ values added in tested tables could be a good identifier when extracting data for final statistical analysis.
• Assists code development more efficiently: the program has been designed from the beginning to the end, to produce the correct and desired results.
• Reusable validated code. This code block can be repeatedly applied as the basic check and also used by all general cases (in our case, used by a majority of studies, such as patient profile, demographic information, vital signs, concurrent medications, safety measurement for adverse events and laboratory data).

CONCLUSION

This presentation has demonstrated a method of designing and constructing a highly structured and clearly-defined validation program. The validation codes I developed can be applied to implementing a data quality assurance procedure for collecting clean clinical trial data in addition to creating the appropriate data files for further statistical analysis. However, it can also be utilized and expanded to your database system.

ACKNOWLEDGMENTS

The author would like to take this opportunity to acknowledge all of the assistance and support given to him by Drs. Fouad Kandeel and Jeff Longmate at City of Hope National Medical Center. The author also expresses his appreciation to his colleague, Lorraine Lesiecki, for correcting the errors during the writing of an earlier draft of this paper.

APPENDIX

```
%MACRO Validation;
  %LOCAL MasterTotalRow;
  %LOCAL WhTotalRow;
  %LOCAL IdTotalRow;
  %LOCAL DataOK;
  PROC SQL NOPRINT;
  DELETE * FROM Mylib.Errors;
  QUIT;
  PROC SQL NOPRINT;
  SELECT COUNT(*) INTO :MasterTotalRow FROM Mylib.Master;
  QUIT;
  %DO I = 1 %TO &MasterTotalRow;
    DATA _NULL_;
    SET Mylib.Master (FIRSTOBS=&I OBS=&I);
    CALL SYMPUT('TableID', Memname);
    CALL SYMPUT('TableName', TblName);
    CALL SYMPUT('WhTblName', WhTblName);
    RUN;
    PROC SQL NOPRINT;
    ALTER TABLE Mylib.&TableID ADD Flag INTEGER;
    QUIT;
    PROC COPY IN=Mylib OUT=Work;
```
SELECT &TableID;
RUN;
PROC SQL NOPRINT;
SELECT COUNT(*) INTO :WhTotalRow FROM Mylib.&WhTblName;
QUIT;
%DO J = 1 %TO &WhTotalRow;
DATA _NULL_;  
SET Mylib.&WhTblName (FIRSTOBS=&J OBS=&J);
CALL SYMPUT('Field', FieldName);
CALL SYMPUT('WhereCondition', WhereCond);
CALL SYMPUT('ErrorMessage', ErrorMsg);
RUN;
%LET wh = WHERE &Field &WhereCondition;
PROC SQL NOPRINT;
SELECT COUNT(*) INTO :IdTotalRow FROM Mylib.&TableID;
QUIT;
%DO K = 1 %TO &IdTotalRow;
DATA Mylib.Temp;
SET Mylib.&TableID;
IF _N_ = &K THEN DO;
CALL SYMPUT('TrackingNo', TRIM(LEFT(PUT(TrackingNo, 8.))));
CALL SYMPUT('varFlag', TRIM(LEFT(PUT(Flag, 8.))));
OUTPUT;
END;
RUN;
PROC SQL NOPRINT;
SELECT COUNT(*) INTO :DataOK FROM Mylib.Temp &wh;
QUIT;
%IF &DataOK > 0 %THEN
%DO;
PROC SQL NOPRINT;
INSERT INTO Mylib.Errors (TrackingNo, RowID, TimeStamp, FormName, TableID, FieldName, ErrorMsg) VALUES(&TrackingNo, &RowID, "&TimeStat", "&TableName", "&TableID", "&Field", "&ErrorMessage");
QUIT;
%END;
%IF &DataOK LE 0 %THEN
%DO;
PROC SQL NOPRINT;
UPDATE Mylib.Temp SET Flag = -1;
QUIT;
%END;
DATA _NULL_;  
SET Mylib.Temp;
CALL SYMPUT('TempFlag', TRIM(LEFT(PUT(Flag, 8.))));
RUN;
DATA Work.&TableID;
SET Work.&TableID;
IF (_N_ = &K AND &TempFlag = -1) THEN Flag = -1;
RUN;
%END;
%END;
%MEND Validation;
%Validation;
CONTACT INFORMATION

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

Keh-Dong Shiang, Ph.D.
Department of Biostatistics & Department of Diabetes
City of Hope National Medical Center
1500 East Duarte Road
Duarte, CA 91010-3000
Work Phone: (626)256-4673  Ext. 65768
Fax: (626)471-7106
E-mail: kshiang@coh.org
Web: www.coh.org

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.