PhUSE 2017

Paper DH06

Constrain your Terminology
Use Integrity Constraints to Validate your Data

Sven Greiner, HMS Analytical Software GmbH, Sulzbach (Taunus), Germany

ABSTRACT

Integrity constraints are sets of data validation rules that restrict the values of a variable in a SAS® dataset. With variable metadata readily available for automated dataset setup, integrity constraints can easily be implemented to validate your data against the corresponding controlled terminology.
Find out about the different types of integrity constraints and how you can use them to integrate data checks into your dataset creation process. Learn how to add integrity constraints to your datasets and the challenges you face when working with these datasets. Let's analyze the pros and cons of using integrity constraints in clinical trial analysis dataset development and compare them to other methods of validating your data.
Integrity constraints may be an easy-to-add feature in your analysis dataset creation process.

INTRODUCTION

This paper demonstrates how you can add integrity constraints (ICs) to your variable definitions to validate the variable values against your pre-specified controlled terminology (CT). The consistency of variable values and CT is a requirement within datasets defined according to the Study Data Tabulation Model (SDTM) or Analysis Data Model (ADaM).
The technical challenges and the (dis)advantages of integrity constraints are considered when assessing the usability of integrity constraints in an analysis dataset (ADS) creation process.

THE BASICS OF INTEGRITY CONSTRAINTS

ICs are a set of data validation rules that restrict the values of a variable in a SAS® dataset. SAS applies these data validation rules whenever records are added, updated, or deleted from the dataset. ICs are classified as general or referential.

GENERAL INTEGRITY CONSTRAINTS

General ICs are data validation rules for a single SAS dataset. There are four types of general ICs:

1. **CHECK**: Limit variable values to a list or range of values. Check constraints can also be used to ensure that data values in one variable within an observation are contingent on the data values in another variable within the same observation.
2. **NOT NULL**: Missing values are not allowed in a variable, a value is required.
3. **UNIQUE**: A unique combination of values is required in the specified variable(s).
4. **PRIMARY KEY**: Unique combination of values in the specified variable(s). Missing values are not allowed. A primary key constraint is a general IC if it is not referenced by a foreign key IC.

REFERENTIAL INTEGRITY CONSTRAINTS

Referencing a primary key IC in one dataset, by a foreign key IC in another dataset creates a referential IC. The foreign key links the data values of one or more variables in the foreign key dataset to corresponding variables in the primary key dataset. The data values in the foreign key dataset must have a matching value in the primary dataset or be missing.
Referential ICs can be defined for update and delete operations as follows:

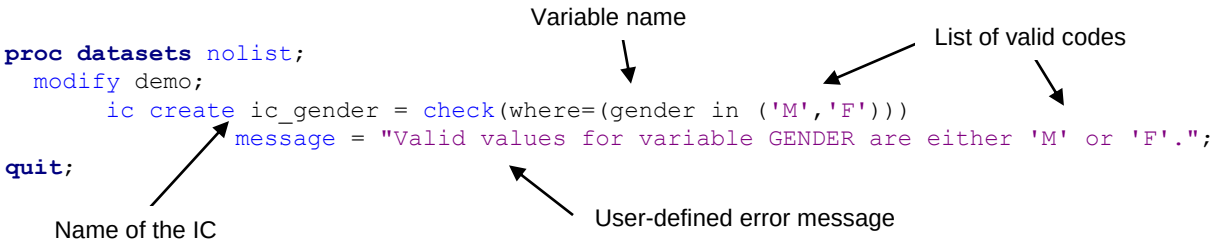
1. **RESTRICT**: Data values of the primary key variables cannot be updated or deleted if a matching foreign key data value exists.
2. **SET NULL**: Data values of the primary key variables may be updated or deleted, but the corresponding foreign key data values are set to missing.
3. **CASCADE**: Data values in the primary key variables may be updated and the foreign keys will be updated to match the values in the primary key.

The use of referential ICs is not in the scope of this paper.

MAKING USE OF INTEGRITY CONSTRAINTS

The essential IC in the scope of this paper is CHECK. The CT codes provided by the ADS specifications, metadata repository or data definition documentation (define.xml) can be used with variable information to create a CHECK IC programmatically. As soon as the IC is applied to the dataset, only values meeting the IC criteria may be added to the dataset.

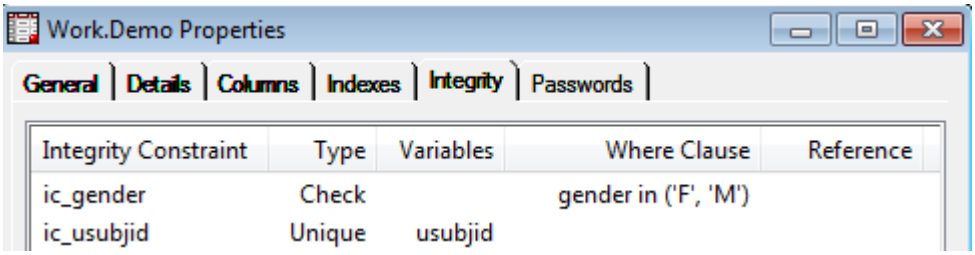
An example of a CHECK IC added to variable GENDER in dataset DEMO using PROC DATASETS:



In addition to the CHECK IC, the NOT NULL and UNIQUE ICs are of interest for the dataset creation process. For certain required ADaM variables no missing values are allowed. Flagging the respective variables in the specifications is the basis for a programmatically created NOT NULL IC to restrict missing values in these variables. Most SAS datasets have a certain combination of variables (called “keys”) that make every observation in the dataset unique. The UNIQUE IC can be created by specifying the key (or list of keys) inside the UNIQUE IC. No duplicate combination of keys may be added to the dataset.

```
proc datasets nolist;
  modify demo;
    ic create ic_usubjid = unique(usubjid);
quit;
```

Ic_gender and ic_usubjid in the dataset properties:



INTEGRITY CONSTRAINTS IN ADS DEVELOPMENT

For efficiency and consistency between specifications, ADS and define.xml, most ADS creation processes utilize the variable metadata directly from the specifications or the define.xml. A standard program reads the required metadata (this usually includes dataset, variable and parameter-level metadata) and creates the code to set up a template dataset. An example of typical ADS specifications is shown below:

Key	Variable Name	Variable Label	Variable Type	Variable Length	Variable Format	Controlled Terminology
1	USUBJID	Unique Subject Identifier	text	20	$20.	
	AGE	Age	num	8		
	SEX	Sex	text	1	$1.	SEXC
	RACE	Race	text	20	$20.	RACEC
	TRTP	Planned Treatment	text	20	$20.	TRTPC

„M“ = „Male“
„F“ = „Female“

„White“ = „White“
„Indian“ = „Indian“
„Black“ = „Black“

„TRT_A“ = „TRT_A“
„TRT_B“ = „TRT_B“
„ “ = „“

The code created from the ADS specifications may look like this:

```
proc sql;
   create table adsl(label='Subject-Level Analysis Dataset')
    (
    usubjid char (20)  label="Unique Subject Identifier" format=$20.,
    age     num   (8)  label="Age",
    sex     char  (1)  label="Sex"  format=$1.,
    race    char (20)  label="Race" format=$20.,
    trtp    char (20)  label="Planned Treatment"  format=$20.
    );
quit;
```

The above PROC SQL creates a template dataset with variable definitions directly from the specifications, but without any content. The content is derived in the ADS creation process and then added to the template. Accordingly, the ADS metadata matches the metadata in the specifications. This type of dataset setup is the basis for many ADS creation processes.

Adding ICs to a template dataset can be done during or after template dataset creation. PROC DATASETS is the only option to add ICs to an existing dataset, but the simplest way is to add IC definitions to the template dataset is PROC SQL. The code below shows how ICs can be defined in the same PROC SQL as the variable definitions:

```
proc sql;
   create table adsl(label='Subject-Level Analysis Dataset')
    (
    usubjid char (20)  label="Unique Subject Identifier" format=$20.,
    …

    constraint CON_SEX check(sex in ('M','F'))
               message = "Valid values for variable SEX: 'M', 'F'",

    constraint CON_RACE check(race in ('White','Indian','Black'))
               message = "Valid values for variable RACE: 'White', 'Indian', 'Black'",

    constraint CON_TRTP check(trtp in ('TRT_A','TRT_B',''))
               message = "Valid values for variable TRTP: 'TRT_A', 'TRT_B', ''",

    constraint UNIQUE_KEYS unique(usubjid));
quit;
```

The name(s) of the variable(s), the IC type, and (if applicable) the respective values of the controlled terminology are matched from the different metadata sources. A concatenation of this information provides the IC definitions shown in the example above.
After executing the code, the new template dataset includes variable definitions and ICs. Adding invalid values to a variable or multiple records with the same USUBJID causes an error in the program. The observation containing the invalid value will not be added to the template dataset.

TECHNICAL CONSIDERATIONS

Working with IC-controlled datasets is quite challenging due to the special handling that is necessary to a) preserve ICs on a dataset when working with it and b) get the desired information if an IC is violated. This section provides an overview of the most relevant challenges.

PRESERVATION OF INTEGRITY CONSTRAINTS

SAS datastep operations are not a useful tool when working with IC-controlled datasets. ICs are only preserved on an existing dataset or copied to a new one with a specific set of procedures. The SAS datastep SET statement is not one of these procedures. Consider the following code:

```
data ads.adsl;
   set template_adsl /* empty template dataset including ICs */
       data_adsl;    /* ADSL data without respective variable metadata and ICs */
run;
```

The result of the above datastep is a dataset ADS.ADSL taking over the variable metadata from the TEMPLATE_ADSL dataset and the data from DATA_ADSL. The ICs are not carried over from TEMPLATE_ADSL to the ADS.ADSL dataset. To achieve this goal, other means of handling the IC-controlled datasets are necessary.

ICs are preserved by the following procedures:

1. PROC SQL: Inserting, updating or deleting observations,
2. PROC APPEND: Appending new data to an IC-controlled base dataset,
3. PROC SORT: Sorting a dataset without using the OUT statement,
4. PROC COPY, CPORT and CIMPORT: Duplicating a dataset if CONSTRAINT=yes is specified.

INSERTING OBSERVATIONS

The PROC SQL INSERT statement is the most efficient way of adding data into an IC-controlled template dataset. The ICs are preserved in the template dataset and every newly added observation is checked against the template dataset ICs before the observation is added. If an IC is violated, the entire INSERT statement is dismissed and the respective error message is written to the log.

The following log code shows how SAS inserts three observation into TEMPLATE_ADSL:

```
52   proc sql;
53      insert into template_adsl
54      values('ABC-001', 55, 'M', 'White',  'TRT_A')
55      values('ABC-002', 26, 'F', 'Blue',   'TRT_B')
56      values('ABC-003', 45, 'M', 'Indian', 'TRT_C');
ERROR: Valid values for variable RACE: 'White', 'Indian', 'Black'
NOTE: This insert failed while attempting to add data from VALUES clause 2 to the data
set.
NOTE: Deleting the successful inserts before error noted above to restore table to a
consistent state.
57   quit;
NOTE: The SAS System stopped processing this step because of errors.
```

The strings marked red show violations of the ICs active in TEMPLATE_ADSL. Note that only the first violation is reported by SAS. The INSERT statement is then disregarded and no data is inserted into TEMPLATE_ADSL. There are two ways to receive a detailed list of IC violations in a dataset:

1. Run, correct and re-run the same INSERT statement over and over again,
2. Insert each observation with an individual INSERT statement.

Considering these two options, the second promises a more efficient ADS creation process. This assumption will come under closer examination in the next section of this paper.

IN CASE OF ERROR

The simplest way of tracking IC violations is via the error messages that are written to the log. These messages can be defined for each IC individually using the MESSAGE statement. The list of valid values is added in the example below to help the programmer identify the invalid value(s). However, the length of the MESSAGE string is restricted to 180 characters, after which the message is truncated. Long or numerous values are likely to cause an incomplete list of values, so the usability of this feature is limited. There is no way of printing the invalid value to the log, making the search for inconsistencies between dataset and CT potentially laborious.

```
constraint CON_RACE check(race in ('White','Indian','Black'))
           message = "Valid values for variable: 'White', 'Indian', 'Black'"
```

```
ERROR: Valid values for variable: 'White', 'Indian', 'Black'. Add/Update failed for
data set WORK.TEMPLATE_ADSL because data value(s) do not comply with integrity
constraint VAL_RACE.
```

The first log message (ERROR:) is specified by the user. The second message (Add/Update failed …) is a SAS-generated message that may be switched off by specifying MSGTYPE=USER.
More detailed information on the IC violations is provided by the use of audit trails, which are discussed in the next section.

PUTTING IT ALL TOGETHER

In the previous sections, this paper has provided an introduction to the benefits and limitations of ICs. The next section combines this information into an example program, which can be used to insert data into an IC-controlled template dataset. Each step in the program is explained and alternative options are discussed. The code does not contain any macro code for better readability. In a real-life scenario this program should be part of a validated standard macro, explicitly developed to check ADS data against its corresponding CT.

Before going through the example code, the following requirements are presented to evaluate whether the code satisfies the needs of an efficient ADS creation process:

1. Multiple datasets run in a row: failure of one ADS shall not compromise the entire run,
2. The CTs shall be checked during ADS program run-time,
3. ADS may or may not have ICs (none, one, multiple),
4. IC violations shall provide the variable and a list of (in-)valid values,
5. The code should be easy to “macro” and quick to run.

INTEGRITY CONSTRAINTS IN ACTION

The following example code provides a best practice of how to use ICs to validate data against the corresponding CT.

```
* Template dataset;
proc sql;
   create table adsl(label='Subject-Level Analysis Dataset')
    (
    usubjid char (20)  label="Unique Subject Identifier" format=$20.,
    age     num   (8)  label="Age",
    sex     char  (1)  label="Sex"  format=$1.,
    race    char (20)  label="Race" format=$20.,
    trtp    char (20)  label="Planned Treatment"  format=$20.

    constraint CON_SEX check(sex in ('M','F'))
               message="Valid values for variable SEX: 'M', 'F'",

    constraint CON_RACE check(race in ('White','Indian','Black'))
               message = "Valid values for variable RACE: 'White', 'Indian', 'Black'",

    constraint CON_TRTP check(trtp in ('TRT_A','TRT_B',''))
               message = "Valid values for variable TRTP: 'TRT_A', 'TRT_B', ''",

    constraint UNIQUE_KEYS unique(usubjid)
    );
quit;
```

The code for the template dataset is programmatically created from the specifications or the define.xml. Section "Integrity Constraints in ADS Development" provides a more detailed explanation of this point.
The template dataset ADSL contains five variables, including USUBJID which has to be unique in the dataset. Variables SEX, RACE and TRTP have a CT assigned to them and the values of these CTs are listed in the corresponding ICs. The dataset does not contain any observations.

```
* Data;
data adsl0;
   length usubjid $20 age 8 sex $1 race $20 trtp $20;
   input usubjid age sex race trtp;
   datalines;
   ABC-001 55 M White  TRT_A
   ABC-002 26 F Blue   TRT_B
   ABC-003 45 M Indian TRT_C;
run;
```

The ADSL0 dataset contains the derived data for the ADSL template dataset. Two of the three observations violate the ICs (values marked red) and will therefore be rejected when inserting ADSL0 into the ADSL template dataset.

```
* Initiate Audit Trail;
proc datasets library=work nolist;
   audit adsl;
   initiate;
quit;
```

The audit trail is an optional SAS file that logs modifications to a SAS dataset. It is also the only feature in SAS that stores observations that were rejected by ICs. The audit trail information is stored in a dataset with the dataset type AUDIT (e.g. ADSL(type=audit)).
The ADSL audit trail is the only way to identify the invalid values from ADSL0 that are rejected when inserting the ADSL0 data into ADSL.

```
proc sql noprint;
   * Number of observations;
   select count(*)
   into :nobs
   from adsl0;

   * List of variables in template dataset;
   select name into :vlist separated by ', '
   from dictionary.columns
   where memname eq upcase("ADSL") and memtype eq "DATA";
quit;
```

Count the number of observations in ADSL0 in macro variable NOBS. This information is necessary to insert each ADSL0 observation individually into ADSL.
Create a list of the ADSL-variables in macro variable VLIST. VLIST determines the variable (and their order) in the INSERT statement.

```
* Insert observations individually;
%macro insert_obs(nobs=);
proc sql;
%do i = 1 %to &nobs.;
   insert into adsl
   select &vlist. from adsl0(firstobs=&i. obs=&i.);
%end;
quit;
%mend insert_obs;

%insert_obs(nobs=&nobs.);
```

```
ABC-001 55 M White  TRT_A
ABC-002 26 F Blue   TRT_B
ABC-003 45 M Indian TRT_C
```

```
NOTE: 1 row was inserted into WORK.ADSL.
ERROR: Valid values for variable RACE: 'White', 'Indian', 'Black'
NOTE: Deleting the successful inserts before error noted above to restore table to a consistent
state.
ERROR: Valid values for variable TRTP: 'TRT_A', 'TRT_B', ''
NOTE: Deleting the successful inserts before error noted above to restore table to a consistent
state.
NOTE: The SAS System stopped processing this step because of errors.
```

The INSERT_OBS macro creates an individual INSERT statement for each observation in ADSL0. Each observation is inserted on its own, resulting in an individual IC check for each observation. If an observation is rejected, only this particular observation is rejected and not the whole ADSL0 dataset.
The first record does not cause a violation of the ICs in the template dataset ADSL. The record is successfully inserted into ADSL. The second and third record both have an invalid value in variables RACE and TRTP respectively. Both observations are rejected and SAS writes the user-defined error message (including the list of valid values) to the log. There is no way to write the invalid values (“Blue” or “TRT_C”) to the log.

The content of ADSL after individually inserting the ADSL0 observations:

	USUBJID	AGE	SEX	RACE	TRTP
1	ABC-001	55	M	White	TRT_A

The main drawback with individually inserting each observation to the template dataset is performance. On a SAS server system, inserting several thousands of observations as outlined is done in a matter of seconds. When testing this approach with bigger datasets, 100.000 records took about 5 minutes of computing time for the INSERT statements alone. Performance makes this approach only worthwhile for smaller datasets.
An alternative approach, with a very high performance, is to insert all observations at once. The section “Inserting Observations” discusses the main disadvantage of this approach: SAS only reports the first IC violation for all observations in the INSERT statement.

```
* Print audit trail;
data audit(drop=_atdatetime_ _atobsno_ _atreturncode_ _atuserid_ _atopcode_);
   set adsl(type=audit);
   where _atopcode_ eq "EA";
run;

proc print data=audit noobs;
   id _atmessage_;
run;
```

The audit trail of the before-mentioned individual INSERT statements is easily transferred into the dataset AUDIT. Where statement _atopcode_ eq "EA" restricts the audit trail data to the part where observations failed to be added to ADSL.
Printing this dataset provides a list of rejected observations and the respective error messages including the list of valid values.

The SAS System

ATMESSAGE	usubjid	age	sex	race	trtp
ERROR: Valid values for variable RACE: 'White', 'Indian', 'Black'	ABC-002	26	F	Blue	TRT_B
ERROR: Valid values for variable TRTP: 'TRT_A', 'TRT_B', ''	ABC-003	45	M	Indian	TRT_C

```
* Output final dataset;
proc datasets nolist;
   modify adsl;
   ic delete _all_;
quit;

proc sql;
   delete from adsl;
   insert into adsl select &vlist. from adsl0;
quit;
```

ADSL contains an incomplete set of data from ADSL0. If IC violations occur two options are available:

1. Report the IC violations and discard ADSL,
2. Report the IC violations and create ADSL without ICs and with a complete set of data.

Option 1 is questionable because a single incorrect CT invalidates the ADS and possibly the entire ADS run. This may result in a considerable increase of work and computing time.
Option 2 is presented in the code above. The PROC DATASETS removes the ICs from the template dataset ADSL. The DELETE statement removes all observations from ADSL and finally, the INSERT statement inserts the ADSL0 data into ADSL without any ICs. The error messages and audit trail output are still available in the log and output file.

SUMMARY

ADSs can readily be checked against the corresponding CT during ADS creation. ICs are a suitable tool for this purpose and with a few extra steps, the derived ADS data is checked against an IC-controlled template dataset.

In case of invalid values, a user-defined error message is written to the log, indicating to every user (programmer, validator, etc.) that data and metadata are inconsistent. This user-defined error message may be set up to include the list of valid values for the variable. The length of the error message is restricted to 180 characters, making it less useful for CTs with long or numerous items.
The only way to identify the invalid value that caused an IC violation is with the means of an audit trail. Via the audit trail data, SAS can provide a report of all rejected observations including the user-defined error message with the concerned variable and the list of valid values. This information is very helpful when updating the CT to comply with the ADS.

A disadvantage of ICs is their behavior after an IC violation is encountered. SAS simply stops any further processing of data and writes only the first violation to the log (and audit trail). This problem can be avoided by inserting each observation individually to the template dataset, but only at the cost of very high computing time. The computing time for several hundreds of thousands of observations may be so high that this approach cannot be recommended. If an observation has multiple IC violations in different variables, there is no workaround to get all violations in one run. Running, correcting and re-running the IC checks is the only way to identify multiple IC violations in a single observation.

After one or multiple IC violations, the derived dataset is empty or incomplete. It is therefore recommended to subsequently re-create the ADS without the use of ICs.

CONCLUSION

ICs offer an easy-to-implement solution to checking your data against your CTs while developing your ADSs. This allows for earlier detection of invalid values (or incomplete CT) compared to other tools that check the CT as one of the last steps in ADS validation (e.g. Pinnacle21, define.xml macros).
However, ICs come with a set of technical limitations that pose a problem when trying to efficiently check data. The need to make a decision between the poor performance of checking each observation individually, and re-running the checks over and over again to identify all IC violations, takes a heavy toll on the usefulness of ICs as a data checking tool.
To avoid the limitations of ICs, some of the alternatives may be considered. Pinnacle21 and some define.xml macros check variable values against the CT. Their execution, however, is an extra step in the ADS creation process, often not taken (or only irregularly). This results in inconsistencies between the ADS and the CT.

An approach that works at run-time of each ADS, but without the limitations of ICs, is the use of a checking macro based on IF or SELECT expressions, which checks the variable values against its list of valid values (SEX not in ('M', 'F') then put …). The code may be as easy to implement as for the ICs, but provides more adaptability, better identification of CT-deviations, and higher performance than the IC approach.

REFERENCES

Integrity Constraints and Audit Trails Working Together, Gary Franklin and Art Jensen
SAS(R) 9.2 Language Reference: Concepts, Second Edition, Understanding Integrity Constraints

CONTACT INFORMATION

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

Author Name: Sven Greiner
Company: HMS Analytical Software GmbH
Address: Otto-Volger-Straße 3c
City / Postcode: 65843 Sulzbach
Work Phone: +49 6221 6051 183
Fax: +49 6221 6051 97
Email: sven.greiner@analytical-software.de
Web: www.analytical-software.de