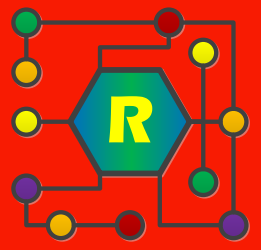
Enforcing Data Integrity with SAS Audit Trails

Vincent Rabatin

PhilaSUG Spring Meeting, June 5, 2019



Rabatin Pharma Services

Data Validation Questions

When we import data from an electronic data capture system or creating an SDTM or AdAM dataset, we want to know if the observations match their specification.

We may ask:

- Do variables have values that are out of range, impermissible or missing?
- If required, do variables correspond to a code list or reference dataset?
- Are dependencies between other variables in the same record enforced?
- Are duplicate records or records that violate primary key constraints present?

Integrity Constraints and Audit Trails

Features introduced in ***SAS*** Versions 7 and 8 can help us answer many data validation questions.

Integrity constraints are rules that place restrictions on the observations that a dataset may contain.

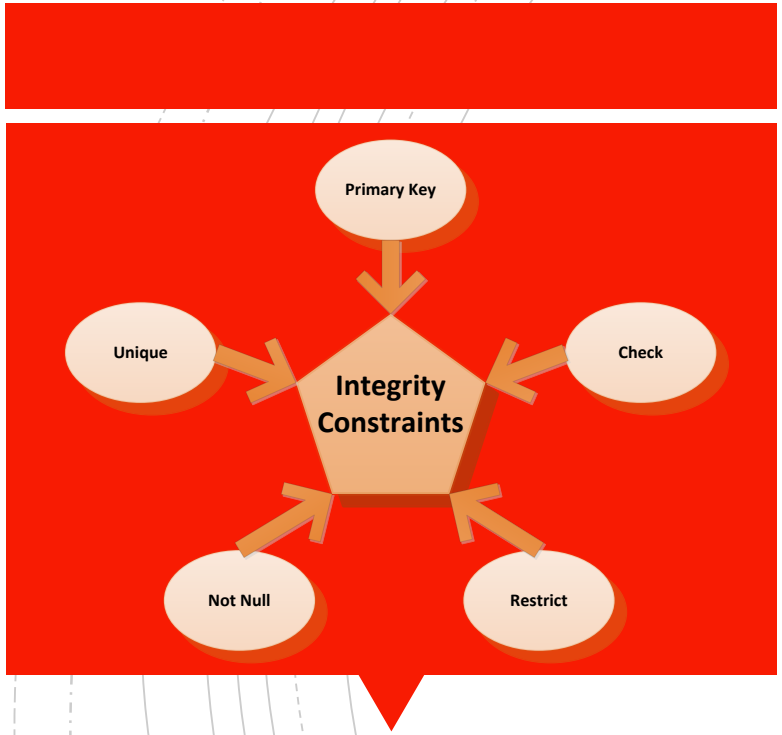
Audit trails log successful and unsuccessful attempts to alter the observations in a dataset.

These features complement each other to ensure that valid observations get passed to the output dataset and the invalid observations get trapped and held for inspection.

Disclaimer

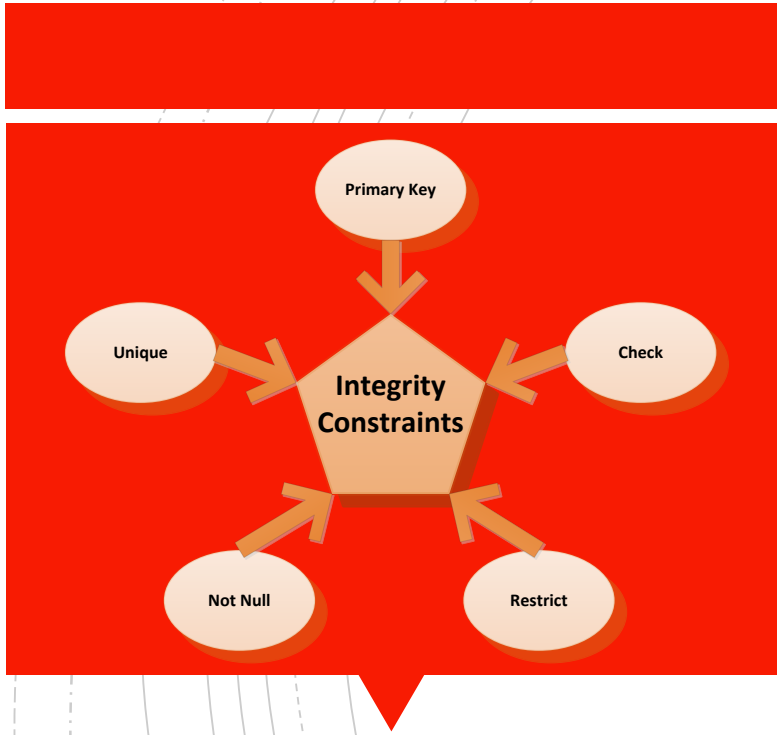
- The methods described here will not detect all types of errors that may occur in the data.
- Constraints on data involving more complicated relations between observations, such as, ensuring that the dates of successive events are greater than those that come before, require additional measures.
- Automated checks are only an aid, there is no substitute for manually checking the dataset before delivery.

Integrity Constraints



- Primary Key – Requires that specified variables(s) be unique and not null.
- Unique – Requires that specified variable(s) be unique: only one null value is allowed.
- Not Null – Requires that the variable contain a value.
- Check – Limits the variable to a specific set, range, or relation to other variables in the same observation.
- Restrict – Does not allow an observation to be inserted unless specified variables match variables in another dataset (foreign key).

Implementing Integrity Constraints



In the ***SAS*** system there are two ways to implement integrity constraints:

- **Proc Datasets**
- **Proc SQL**

Which you choose is a matter of preference. My purpose is to encourage you to automate the process by developing a macro driven by the metadata of your project.

Driven by the project's metadata, have your macro create an empty dataset whose variable attributes correspond to the specification. It might produce code like this.

```
Proc SQL;
    create table mystudy.subjects
    (SUBJID char(10) label="Subject Identifier",
    SEX char(1) label="Gender",
    AGE num label="Age in Years",
    SITE char(3) label="Site ID",
    SITENAME char(35) label="Site Name")
;
```

Metadata for constructing the target datasets

- DATASET (to be created)
- NAME
- TYPE
- LENGTH
- LABEL
- ORDER
- FORMAT (optional)

If your project contains a lengthy code list you may store it in an auxiliary dataset. This one is constrained by a primary key: it will therefore contain an index.

```
/*** the sitelist dataset will serve as a lookup table ***/

create table mystudy.sitelist
    (SITE char(3),
    SITENAME char(35),
/* This is how you add a constraint in PROC SQL */
    constraint mylist primary key(SITE, SITENAME))
;
```

Let's fill the lookup table with the valid site code and names used in our project.

```
/* load the lookup table with valid sites */
insert into mystudy.SITELIST(SITE, SITENAME)
values("001","Smith Therapy Associates")
values("002","Maximus Clinic")
values("003","Jones Hospital")
values("004","Rodgers Oncology Center")
values("005","Feldspar Group")
values("006","United Physicians Treatment Center")
;
quit;
```

Primary Key

Check: Value in list

Check: Value in range

Foreign Key: Values match those in the lookup dataset

```
/*** Apply Constraints with PROC DATASETS to the subjects table ***/
/*** using PROC DATASETS ***/

proc datasets nolist library=mystudy;
 modify subjects;
     /* Set the primary key (this also creates an index on the dataset) */
     ic create pk = PRIMARY KEY(SUBJID)
          message= "Only one record per subject is allowed.";

     /* Check for valid values by list */
     ic create val_sex = CHECK(where=(sex in ('M','F')))
          message = "Valid values for variable SEX are either 'M' or 'F'.";

     /* Check for valid values by computation */
     ic create val_age = CHECK(where=(18 <= age <= 55))
          message = "An invalid AGE has been provided.";

     /* Check for valid values by key in another dataset */
     ic create site_list=FOREIGN KEY (SITE SITENAME)
      REFERENCES mystudy.SITELIST;

run;
quit;
```

A CHECK constraint may be used to validate variables dependent on others in the same observation. For example: A dataset may contain two variables: ***reason*** and ***other_reason***. Codes 1 through 4 map to a list of expected reasons. The code for **"Other Reason"** is 5. If variable ***reason*** is not 5, then ***other_reason*** must be blank. If ***reason*** is 5, then ***other_reason*** must contain text.

```
/* Check for relations between variables */
    ic create val_reason_other = CHECK(where=((reason in(1,2,3,4)
        and other_reason="") or (reason=5 and other NE ""))
        message = "If reason is 5, other_reason cannot be blank, else other_reason
    must be blank.";
```

Metadata for constructing the constraints

- DATASET
- CONSTRAINT_NAME
- CONSTRAINT_TYPE
- CONSTRAINT_TEXT
- CONSTRAINT_MESSAGE

If you prefer PROC DATASETS to PROC SQL, your macro might contain a code fragment such as this.

```
%do i=1 to &number_of_constraints;
   ic create &&constraint_name(&i)= &&constraint_type(&i)
      (&&constraint_text(&i))
      message = "&&constraint_message(&i)";
%end;
```

Similarly, the code for PROC SQL would be:

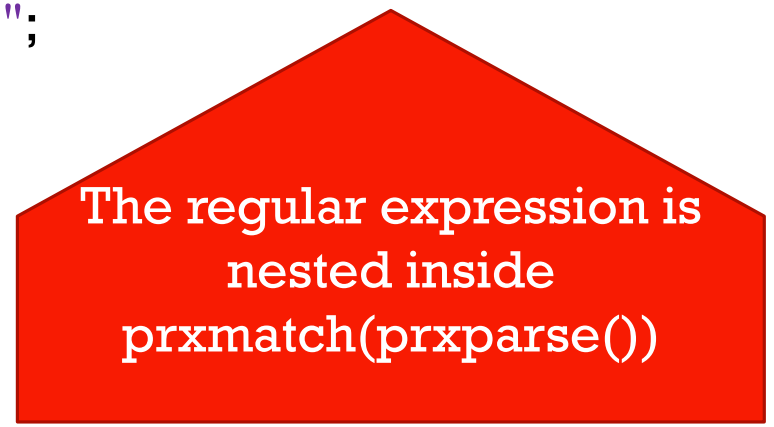
```
%do i=1 to &number_of_constraints;
    constraint &&constraint_name(&i)  &&constraint_type(&i)
        (&&constraint_text(&i))
        message = "&&constraint_message(&i)"
%end;
```

Using Regular Expressions to Validate Variables (1)

You may ask if regular expressions may be used to validate variables. The answer is yes. In our sample study, the variable ***subjid*** has the form **SUBJ-ddddd**, where the d's are digits.

```
/*** Add a constraint to validate subject */
proc datasets nolist library=mystudy;
  modify mystudy;
      ic create val_subj = check(where=(0 < prxmatch(prxparse("/^SUBJ-[0-9]{5}/ "), subjid)))
            message = "An invalid SUBJECT has been provided.";
run;

Quit;
```



Using Regular Expressions to Validate Variables (2)

What if the regular expression is really long, or we want to reuse it for several other variables, such as ISO8601 datetimes?
If that's the case, we can declare the regular expression as a macro variable.
Here, I use SQL syntax to declare the constraint.

```
%let ISO=/^(-?(?:[1-9][0-9]*)?[0-9]{4})-(1[0-2]|0[1-9])-(3[0-1]|0[1-9]|[1-2][0-
9])T(2[0-3]|[0-1][0-9]):([0-5][0-9]):([0-5][0-9])(\.[0-9]+)?$/;

Proc SQL;
create table DATELIST
(ISODATE char(40),
constraint ISOCHK check (0 < prxmatch(prxparse("&ISO"), ISODATE)));
;
```

Audit Trail

The next step is to apply an audit trail to trap records that violate the integrity constraints. The following slides explain how the process works.

A SAS Audit Trail:

- Is of type *audit* with the same name as the parent SAS data file
- Logs modifications to a SAS data file
  - Additions
  - Deletions
  - Updates
- Provides tracking information
  - Who made the modification
  - What was modified
  - When it was made
  - Why an operation failed!

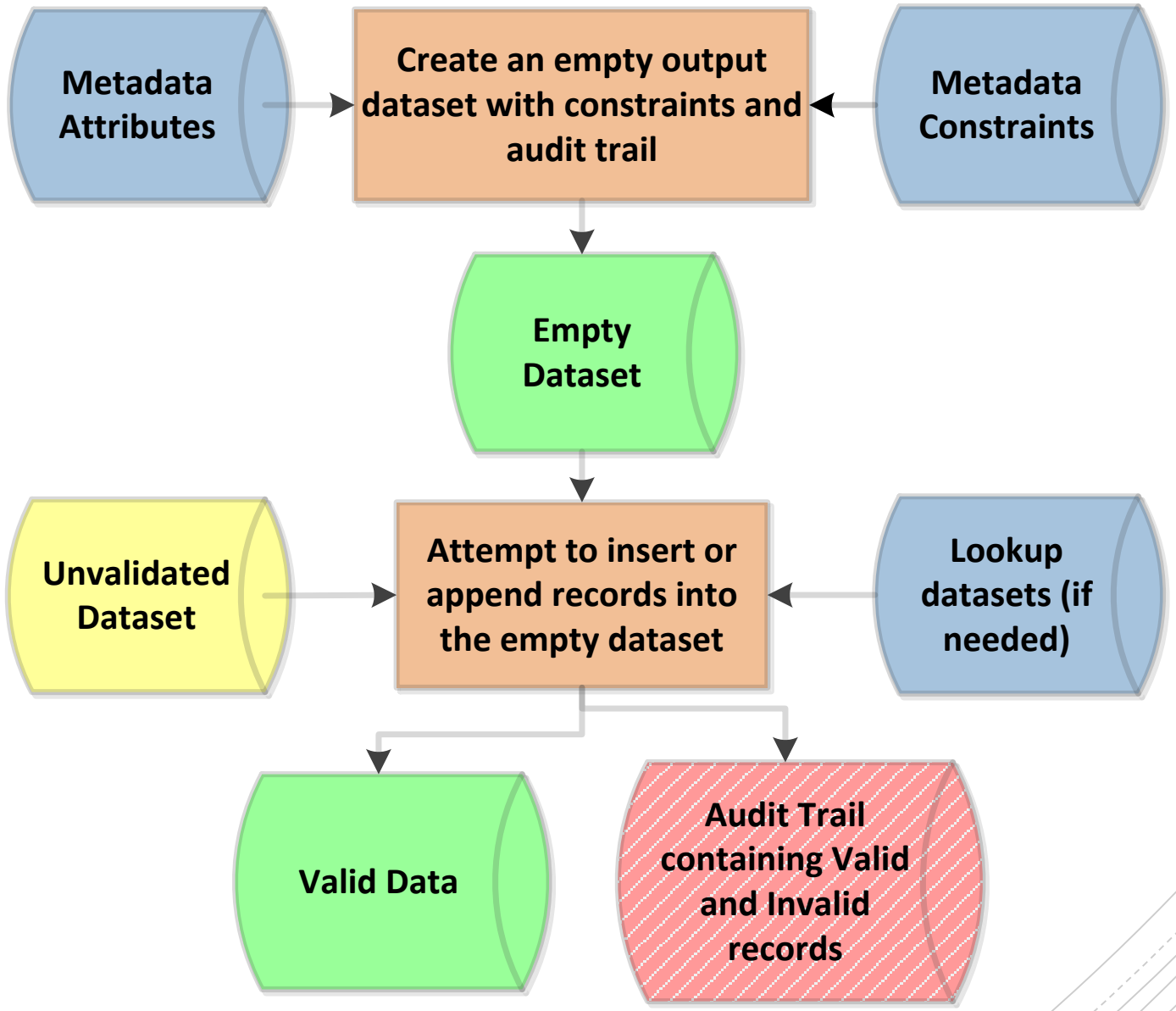
How to Initiate an Audit Trail

```
/*** Use PROC DATASETS to initiate an audit trail ***/

Proc Datasets nolist library=mystudy;
     audit mystudy.subjects;
     initiate;
     log error_image=yes;
Run;

Quit;
```

Strategy of the Macro



Test Data with Errors

```
/*** Create a test dataset, subjects_raw ***/

data subjects_raw;
input SUBJID $10. +1
      SEX $1. +1
      AGE 2. +1
      SITE $3. +1
      SITENAME $35.
;
datalines;
SUBJ-00001 M 55 003 Jones Hospital
SUBJ-00002 M 05 006 United Physicians Treatment Center
SUBJ_00003 F 23 003 Jones Hospital
SUBJ-00004 U 42 003 Jones Hospital
SUBJ-00007 F 23 003 Jonas Hospital
SUBJ-00011 M 55 005 Feldspar Group
SUBJ-00017 F 37 003 Jones Hospital
SUBJ-00018 F 75 001 Smith Therapy Associates
SUBJ-00022 M 15 003 Jones Hospital
SUBJ-00001 M 31 003 Jones Hospital
;
run;
```

Append records to the dataset

```
PROC APPEND base=mystudy.subjects
    data=subjects_raw;
run;
```

Checking the Results

1. The audit trail will contain an operation code _AEOPCODE_ for all attempted records. An _AEOPCODE_ of "EA" signifies a failure of the record to be added. _ATMESSAGE_ contains the message we assigned. We will print records from the audit trail with an "EA" opcode.
2. The SAS Log of PROC APPEND will reveal mismatches between the length of variables on the input dataset and the length of variables on the standardized dataset.

Print any Errors

```
Proc print data=mystudy.subjects (type=audit);
var SUBJID SEX AGE SITE SITENAME
    _ATMESSAGE_;
where _ATOPCODE_="EA";
/* Code EA is for Observation Add Failed */
run;
```

Obs	SUBJID	SEX	AGE	SITE	SITENAME	_ATMESSAGE_
1	SUBJ-00002	M	5	006	United Physicians Treatment Center	ERROR: An invalid AGE has been provided. Add/Update failed for data set MYSTUDY.SUBJECTS because data value(s) do not comply with integrity constraint val_age.
2	SUBJ_00003	F	23	003	Jones Hospital	ERROR: An invalid SUBJECT has been provided. Add/Update failed for data set MYSTUDY.SUBJECTS because data value(s) do not comply with integrity constraint val_subj.
3	SUBJ-00004	U	42	003	Jones Hospital	ERROR: Valid values for variable SEX are either 'M' or 'F'. Add/Update failed for data set MYSTUDY.SUBJECTS because data value(s) do not comply with integrity constraint val_sex.
4	SUBJ-00007	F	23	003	Jonas Hospital	ERROR: Observation was not added/updated because a matching primary key value was not found for foreign key site_list.
5	SUBJ-00018	F	75	001	Smith Therapy Associates	ERROR: An invalid AGE has been provided. Add/Update failed for data set MYSTUDY.SUBJECTS because data value(s) do not comply with integrity constraint val_age.
6	SUBJ-00022	M	15	003	Jones Hospital	ERROR: An invalid AGE has been provided. Add/Update failed for data set MYSTUDY.SUBJECTS because data value(s) do not comply with integrity constraint val_age.
7	SUBJ-00001	M	31	003	Jones Hospital	ERROR: Only one record per subject is allowed. Add/Update failed for data set MYSTUDY.SUBJECTS because data value(s) do not comply with integrity constraint pk.

Cleanup 1: Remove the audit trail

```
Proc Datasets nolist
library=mystudy;
        audit subjects;
        terminate;
Run;
Quit;
```

Cleanup 2 – Remove the constraints

If the constraints are not removed, the output dataset cannot be deleted or overwritten. If there are errors, you must remove the constraints before reconstructing it.

```
proc datasets nolist library=mystudy;
  modify subjects;
         ic delete pk;
         ic delete val_sex;
         ic delete val_age;
         ic delete site_list;
         ic delete val_subj;
run;
```

Discussion

These techniques can go a long way to ensuring that the output datasets are correct, but errors have a way of creeping in. The quality of the constraints we apply is dependent on our imagination. Will we think of everything that can go wrong? Are there problems that could occur that cannot be caught by the constraints we put on the datasets? Of course. So additional quality control measures, such as double programming and visual checks are still needed.

References

Integrity Constraints and Audit Trails Working Together
Gary Franklin, SAS Institute Inc, Austin, TX
Art Jansen, SAS Institute Inc, Englewood, CO

https://support.sas.com/resources/papers/proceedings/proceedings/sugi25/25/aa/25p008.pdf

https://support.sas.com/documentation/cdl/en/lrcon/62955/HTML/default/viewer.htm#a001224397.htm

https://mafiadoc.com/audit-trails-for-sas-data-sets_59f356db1723dd89e74bc7f5.html