Integrity Constraints and Audit Trails Working Together

Gary Franklin, SAS Institute Inc., Austin, TX
Art Jensen, SAS Institute Inc., Englewood, CO

ABSTRACT

New features in Version 7 and Version 8 of the SAS ® System support centralized control of updating. The integrity constraints feature defines rules to be enforced during updates, and the audit trail feature logs every update activity, with enough information to determine which updates failed, if any.

When updates fail, data from the audit trail can be used to reconstruct the original updates so they can be successfully reapplied. Using these new features together can substantially reduce the amount of coding required to perform the repair operations.

INTRODUCTION

The purpose of this paper is to introduce the integrity constraints and the audit trail features, and to illustrate how they can be used together to better control and track updates to SAS data files.

INTEGRITY CONSTRAINTS

Integrity constraints are a set of data validation rules that you can specify to restrict the data values accepted into a SAS data file. The integrity constraint rules are enforced whenever values associated with an integrity constraint variable are added, updated, or deleted. Using integrity constraints helps preserve the correctness and consistency of the stored data.

There are two categories of constraints – general and referential.

GENERAL INTEGRITY CONSTRAINTS

General integrity constraints allow you to restrict the data values of variables within a single data file. There are four types of general integrity constraints:

Primary Key — requires that the specified variable(s) contain unique values. Null (or missing) data values are not allowed. Only one primary key can exist in a data file.

NOTE: A primary key is a general integrity constraint as long as it does not have any foreign keys referencing it.

Unique — requires that the specified variable(s) contain unique data values. A null data value is allowed but is limited to a single instance, given the unique nature of the constraint.

Not Null — requires that a variable contain a data value. Null data values are not allowed. Not null constraints are limited to a single variable, requiring a separate integrity constraint be defined for each variable you do not want to allow null values.

Check — limits the data values of variables to a specific set, range, or list 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.

To illustrate how easy it is to define a constraint, suppose you are responsible for maintaining a SAS data file with the names of all the drivers who have ever driven in the Indianapolis 500. The data file contains three columns: the driver's name (Name), how many times they have driven in the race (Starts), and how many times they have won the race (Wins). The first four observations of the DRIVERS data file are shown below:

Name	Starts	Wins
A.J. Foyt Jr.	35	4
Mario Andretti	29	1
Al Unser	27	4
Gordon Johncock	24	2

The driver's name must be unique and cannot be null, thus making it a good candidate for a primary key. The number of starts must be one or greater, which can be enforced with a check constraint. To create the primary key and the check constraints, you can submit the following PROC DATASETS code:

```
proc datasets library=Indy;
  modify Drivers;
    ic create Driver = primary key(Name);
    ic create Starts = check(where=
       (Starts >= 1));
  run;
quit;
```

Prior to creating the primary key constraint, the software verifies that each value for column 'Name' is unique, and is not null. If for some reason these conditions are not met, an error message is displayed indicating the constraint was not created. The software also verifies that each value for column 'Starts' is greater than or equal to one.

Once the constraints are created, PROC CONTENTS can be used to display descriptive information about the constraints and the indexes they utilize. The constraint and index information for the INDY.DRIVERS data file appears below:

```
---Alphabetic List of Integrity Constraints----

    Integrity                          Where
 #  Constraint Type        Variables  Clause
 ----------------------------------------------
 1  Driver     Primary Key  Name
 2  Starts     Check                  Starts>=1


---Alphabetic List of Indexes and Attributes---

                                       # of
             Unique   Built    Owned   Unique
 #    Index  Option   by IC    by IC   Values
 ---------------------------------------------
 1    Name     YES     YES      YES       654
```

Notice in this example that the name assigned to the primary key constraint differs from the column name. Integrity constraint names are not required to match their column name.

Beginning with Version 8, you can use the PROC CONTENTS OUT2= option to store a data file’s integrity constraint and index information in a separate SAS data file. The corresponding PROC DATASETS commands that can be used to re-create the integrity constraints and indexes are also stored as part of the OUT2= data file.

If an attempt is made to add an observation to data file INDY.DRIVERS and the data violates one of the defined integrity constraints, an error message is displayed. For example, the following SQL code tries to add an existing name to the DRIVERS data file:

```
proc sql;
  insert into Indy.Drivers
    values(‘Al Unser’, 1, 0);
quit;

ERROR: Add/Update failed for data set
       INDY.DRIVERS because data value(s) do
       not comply with integrity constraint
       Driver.
```

The error message indicates the input data did not comply with integrity constraint ‘Driver’.

If you would like to define your own integrity constraint error message, you may do so by using the MESSAGE= option of the IC CREATE statement. The previous example, where the ‘Driver’ constraint was defined, can be replaced with the following code:

```
proc datasets library=Indy;
   modify Drivers;
     ic create Driver = primary key(Name)
       message=”Driver name already exists.”;
     ic create Starts = check(where=
       (Starts >= 1));
   run;
quit;
```

The text supplied with the MESSAGE= option will be added to the beginning of the SAS error message. Submitting the same SQL INSERT statement as before will now generate the following error message:

```
ERROR: Driver name already exists.
       Add/Update failed for data set
       INDY.DRIVERS because data value(s) do
       not comply with integrity constraint
       Driver.
```

The user-defined portion of the error message and the SAS portion are separated with a blank character. If you like, you can suppress the SAS portion of the error message by including the MSGTYPE= option, like so,

```
proc datasets library=Indy;
   modify Drivers;
     ic create Driver = primary key(Name)
       message=”Driver name already exists.”
       msgtype=user;
     ic create Starts = check(where=
       (Starts >= 1));
   run;
quit;
```

Submitting the same SQL INSERT statement once more will generate this error message:

```
ERROR: Driver name already exists.
```

NOTE: The MSGTYPE= option was implemented in Release 8.1 of the SAS system.

Suppose that in addition to maintaining the INDY.DRIVERS data file, you are also responsible for maintaining a SAS data file containing the names of drivers who have won the Indianapolis 500. The first four observations of the INDY.CHAMPIONS data set are listed below:

Name	Year
Ray Harroun	1911
Joe Dawson	1912
Jules Goux	1913
Rene Thomas	1914

In order to have won the Indianapolis 500 you must have driven in it. Therefore, the DRIVERS and CHAMPIONS data files can be related to each other through the creation of a referential integrity constraint.

REFERENTIAL INTEGRITY CONSTRAINTS

Referential integrity constraints are created when a primary key constraint in one data file is referenced by a foreign key constraint in another data file. The foreign key constraint links the data values of one or more variables in the foreign key data file to corresponding variables and values in the primary key data file. Data values in the foreign key data file must have a matching value in the primary key data file, or they must be null. Multiple occurrences of the same foreign key values share a common parent record in the primary key data file. When data is updated or deleted in the primary key data file, the modifications are controlled by a referential action, defined as part of the foreign key constraint. Separate referential actions can be defined for the update and delete operations. There are three types of referential actions:

Restrict prevents the data values of the primary key variables from being updated or deleted if there is a matching data value in one of the foreign key data files’ corresponding foreign key variables. The restrict referential action is the default action if one is not specified.

Set Null allows the data values of the primary key variables to be updated or deleted, but matching data values in foreign key data files are set to null (missing) values.

Cascade allows the data values in the primary key to be updated, and additionally updates matching data values in foreign key data files to the same values as those of the primary key.

> NOTE: The cascade referential action was implemented in Release 8.1 of the SAS system, and applies only to update operations, NOT deletes.

The foreign key and its referenced primary key are the two halves that make up a referential integrity constraint. The foreign key data file and its referenced primary key data file can reside in the same SAS library or in different libraries. For referential integrity constraints to be created the primary key and foreign key must:

- contain the same number of variables
- the variables must be of the same type and length
- the variables must be referenced in the same order, and

- if the foreign key is being added to a data file that already contains data, the data values in the foreign key must match existing values in the primary key or be null.

To create a foreign key for column ‘Name’ in the INDY.CHAMPIONS data file that is linked to column ‘Name’ in the INDY.DRIVERS data file, submit the following code:

```
proc datasets library=Indy;
  modify Champions;
    ic create Champion = foreign key(Name)
      references Indy.Drivers
      on update cascade on delete restrict;
  run;
quit;
```

The “on update” and “on delete” portions of the IC CREATE statement are optional. However, if no referential actions are specified then the default update and delete actions are set to restrict.

NOTE: There is no limit to the number of foreign keys that can reference a primary key. However, additional foreign keys adversely impact the performance of update and delete operations.

A third and final data file that you are responsible for maintaining contains information about the teams that entered the 1999 Indianapolis 500. The data file contains five columns, which include the team’s name (Entrant), the team’s driver (Driver), the team’s car number (Car), and the type of engine (Engine) and tires (Tires) used. The first four observations of the INDY.ENTRANTS data file are:

Entrant	Driver	Car	Engine	Tires
Team Menard, Inc.	Greg Ray	2	Oldsmobile Aurora	Firestone
Brant Motorsports	Raul Boesel	3	Oldsmobile Aurora	Goodyear
Panther Racing, LLC	Scott Goodyear	4	Oldsmobile Aurora	Goodyear
Treadway Racing, LLC	Arie Luyendyk	5	Oldsmobile Aurora	Firestone

There are several columns in the ENTRANTS data file that would benefit from integrity constraints. An entrant’s name cannot be null, which can be enforced with a not null constraint. A unique constraint can be defined for column ‘Car’, since each car must have a unique number for quick identification during the race. There are also a limited number of qualified suppliers of the engines and tires used in the race. Defining check constraints for the ‘Engine’ and ‘Tires’ columns will ensure that only the names of the qualified suppliers can be entered into the data file. The qualified suppliers of the engine are the Oldsmobile Aurora and the Nissan Infiniti. The qualified tire suppliers are Firestone and Goodyear. To define these constraints, submit the following code:

```
proc datasets library=Indy;
  modify Entrants;
    ic create Entrant = not null(Entrant);
    ic create Number = unique(Car);
    ic create Engine = check(where=
      ((Engine = ‘Oldsmobile Aurora’) or
      (Engine = ‘Nissan Infiniti)));
    ic create Tires = check(where=
      (Tires in (‘Firestone’ ‘Goodyear’)));
  run;
quit;
```

We have now created three data files that pertain to the Indianapolis 500 (DRIVERS, CHAMPIONS, and ENTRANTS), and one of each type of integrity constraint. The integrity constraints could also have been created using the SQL procedure or SCL functions. In addition to controlling updates to these data files through the use of integrity constraints, the updates can be monitored and analyzed with the audit trail feature.

AUDIT TRAIL

The audit trail is an optional SAS file that logs modifications to a SAS data file. Each time an observation is added, deleted, or updated, information is written to the audit trail about who made the modification, what was modified, and when the modification took place. The audit trail maintains historical information about the data, which gives you the opportunity to develop usage statistics and patterns. The historical information allows you to track individual pieces of data from the moment they enter the data file to the time they leave. The audit trail is also the only facility in the SAS System that stores observations from failed append operations, or that were rejected by integrity constraints.

To illustrate how easy the audit trail is to use, the following statements initiate an audit trail for data file INDY.CHAMPIONS:

```
proc datasets library=Indy;
  audit Champions;
    initiate;
    log before_image=yes data_image=yes
      error_image=yes;
  run;
quit;
```

Once initiated, PROC CONTENTS can be used to display whether the audit trail is active, and what events are being logged. For example:

```
proc contents data=Indy.Champions;
run;
```

generates the following output, which contains information about the audit trail:

```
               The CONTENTS Procedure

Data Set Name: INDY.CHAMPIONS Observations:  90
Member Type:   DATA           Variables:     2
Engine:        SASE7          Indexes:       1
Created:       DDMMMYY        Constraints:   1
Last Modified: DDMMMYY        Obs Length:    40
Protection:                   Deleted Obs:   0
Data Set Type:                Compressed:    NO
Label:                        Sorted:        NO
Audit:              Active
Audit Before Image: YES
Audit Error Image:  YES
Audit Data Image:   YES
```

The last four lines of the output indicate the audit trail is active, and the BEFORE_IMAGE, ERROR_IMAGE and DATA_IMAGE events are currently being logged. The default behavior is to log all three types of events. You can turn off logging of any image with the LOG statement.

SUSPEND, RESUME, and TERMINATE statements are available for temporarily suspending and for deleting the audit trail.

The audit trail is created by the base engine and has the same libref and member name as the data file, but has a data type of AUDIT. It replicates the columns in the data file, and additionally stores two types of special columns:

- _AT *_ columns, which automatically store modification information
- user columns, which are optional columns you can define when you initiate the audit trail.

A description of the _AT *_ columns are included in the following table:

_AT *_ COLUMNS	DESCRIPTION
ATDATETIME	Date and time of the modification
ATUSERID	Logon userid associated with the modification
ATOBSNO	Observation number affected by the modification, except when REUSE=YES (because the observation number is always 0)
ATRETURNCODE	Event return code
ATMESSAGE	SAS log message at the time of modification
ATOPCODE	Operation code describing the type of modification

The _ATOPCODE_ values are listed in the following table:

OPERATION CODE	TYPE OF MODIFICATION
DA	Added data record image
DD	Deleted data record image
DR	Before-update record image
DW	After-update record image
EA	Observation add failed
ED	Observation delete failed
EW	Observation update failed

The types of entries stored into the audit trail file, along with their corresponding _ATOPCODE_ values, are determined by the options specified on the LOG statement when the audit trail is initiated. The "E" operation codes are controlled by the ERROR_IMAGE option. The "DR" operation code is controlled by the BEFORE_IMAGE option, and all other "D" operation codes are controlled by the DATA_IMAGE option. If the LOG statement is omitted when the audit trail is initiated, the default setting for all three images is YES.

The user variable is a variable that associates data values with the data file without making them part of the data file. That is, the data values are stored in the audit file, but you update the variable in the data file like any other data file variable. You may want to define a user variable to enable end users to enter a reason for each update.

User variables are defined at audit trail initiation with the USER_VAR statement. The following code initiates an audit trail and creates a user variable for data files INDY.CHAMPIONS and INDY.DRIVERS. The CHAMPIONS data file's existing audit trail, which was created in the previous example, is terminated prior to initiating the new audit trail:

```
proc datasets library=Indy;
   audit Champions;
      terminate;
      initiate;
      user_var Reason_user_var $ 25;
   run;

   audit Drivers;
      initiate;
      user_var Reason $ 25
         label = "Reason for Update";
   run;
quit;
```

Once initiated, the base engine retrieves the user variables from the audit trail and displays them when the data file is opened for update. You can enter data values for user variables just as you would for any data file variable. The data values are saved to the audit trail as each observation is saved. The user variables are not available when the data file is opened for browsing or printing. However, to rename a user variable or modify its attributes, you must modify the data file, not the audit file. The following example uses PROC DATASETS to rename the user variable:

```
proc datasets library=Indy;
   modify Champions;
      rename Reason_user_var = Reason;
   run;
quit;
```

The audit trail is read-only. It can be read by any component of the SAS System that reads a data set. For example, to view the contents of the INDY.CHAMPIONS data file's corresponding audit trail, use the TYPE= option with a value equal to AUDIT as follows:

```
proc contents data=Indy.Champions
   (type=audit);
run;
```

The output is shown below. Notice that the audit trail contains all the variables from its corresponding data file, the _AT*_ variables, and the user variable.

```
              The CONTENTS Procedure

Data Set Name: INDY.CHAMPIONS Obs:           0
Member Type:   AUDIT          Variables:     9
Engine:        SASE7          Indexes:       0
Created:       DDMMMYY        Obs Length:    129
Last Modified: DDMMMYY        Deleted Obs:   0
Protection:                   Compressed:    NO
Data Set Type: AUDIT          Sorted:        NO
Label:

--Alphabetic List of Variables and Attributes--

 #  Variable       Type Len  Pos    Format
-----------------------------------------------
 1  Name           Char  30    8
 3  Reason         Char  25   38
 2  Year           Num    8    0
 4  _ATDATETIME_   Num    8   63  DATETIME19.
 9  _ATMESSAGE_    Char   8  121
 5  _ATOBSNO_      Num    8   71
 8  _ATOPCODE_     Char   2  119
 6  _ATRETURNCODE_ Num    8   79
 7  _ATUSERID_     Char  32   87
```

NOTE: Updates to the data file are also written to the audit file, which can adversely impact system performance.

CONSTRAINTS AND AUDIT TRAIL TOGETHER

To illustrate how integrity constraints and the audit trail can work together, we will add the list of drivers who drove in the 1999 Indianapolis 500 (INDY.ENTRANTS) to the list of all drivers who have ever driven in the race (INDY.DRIVERS). This will add a new record to the DRIVERS data file for each rookie that drove in the 1999 race. All other observations will be rejected by primary key ‘Driver’. Both the observations that are successfully added and those that are rejected by an integrity constraint will be logged on the audit trail. To append the names of the 1999 drivers to the DRIVERS data file, we will create an interim data file named WORK.TEMPORARY. To help analyze the contents of the audit trail, we will supply a value for the ‘Reason’ user variable previously defined in the DRIVERS audit file. Notice that the reason code to be logged into the audit trail is created as part of the temporary data file:

```
    data work.temporary;
       set Indy.Entrants;
       drop Entrant Car Engine Tires;

       length Reason $ 25;

       rename Driver = Name;
       Starts = 1;
       Wins = 0;
       Reason = 'Add 1999 Rookies';
    run;
```

Once the temporary file has been created, it can be appended to the INDY.DRIVERS data file. Doing so results in the following output:

```
 proc append data=work.temporary
    base=Indy.Drivers;
 run;
 NOTE: Appending WORK.TEMPORARY to INDY.DRIVERS.

 WARNING: Driver name already exists. (Occurred
          29 times.)
 NOTE: 4 observations added.
 NOTE: The data set INDY.DRIVERS has 658
       observations and 4 variables.
```

The names of four rookies have been added to the INDY.DRIVERS data file. Twenty-nine observations were rejected by primary key ‘Driver’ because the names of those drivers already existed in the INDY.DRIVERS data set. To print the audit trail observations that have the ‘Added data record image’ value (DA) in the ‘_ATOPCODE_’ variable, submit the following code:

```
   proc print data=Indy.Drivers (type=audit
      keep=Name Reason _atopcode_ _atmessage_);
      where _atopcode_ = ‘DA’;
   run;
```

The names of the rookie drivers that were added are:

Name	Reason	_ATMESSAGE_
John Hollansworth Jr.	Add 1999 Rookies	
Wim Eyckmans	Add 1999 Rookies	
Robby McGehee	Add 1999 Rookies	
Jeret Schroeder	Add 1999 Rookies	

To view the rejected observations, submit the following statements:

```
   proc print data=Indy.Drivers (type=audit
      keep=Name Reason _atopcode_ _atmessage_);
      where _atopcode_ = ‘EA’;
   run;
```

The first five rejected observations in the INDY.DRIVERS audit trail are:

Name	Reason	_ATMESSAGE_
Greg Ray	Add 1999 Rookies	ERROR: Driver name already exists.
Raul Boesel	Add 1999 Rookies	ERROR: Driver name already exists.
Scott Goodyear	Add 1999 Rookies	ERROR: Driver name already exists.
Arie Luyendyk	Add 1999 Rookies	ERROR: Driver name already exists.
Eliseo Salazar	Add 1999 Rookies	ERROR: Driver name already exists.

The ‘_ATMESSAGE_’ variable for each observation contains the user-defined integrity constraint message that we defined in the earlier example.

Now that the rookie drivers have been added to the INDY.DRIVERS data file, we must increment the ‘Starts’ variable for each of the veteran drivers to indicate they drove in the 1999 Indianapolis 500. We can use the rejected observations from the audit trail to accomplish this task. First, we will create a temporary data file containing the name of each driver who was not added to the INDY.DRIVERS data file by reading the records in the audit trail that have the ‘Observation add failed’ value (EA) in the ‘_ATOPCODE_’ variable. Then, we will use the SQL procedure to increment the ‘Starts’ value for names in INDY.DRIVERS that have a match in the temporary data file. The code to perform these tasks is shown below:

```
   data work.temporary;
      set Indy.Drivers (type=audit
         where=(_atopcode_ = 'EA'));
      keep Name;
   run;

   proc sql;
      update Indy.Drivers
         set Starts = Starts + 1,
             Reason = “Add 1999 Start”
         where Name in (select Name from
            work.temporary);
   quit;
```

The 29 observations containing the drivers’ names that have already driven in the race had their ‘Starts’ variable incremented by one.

In addition to updating the ‘Starts’ column for each of the veteran drivers, we must increment the ‘Wins’ column of the driver who won the 1999 race. The following code performs this task:

```
   proc sql;
      update Indy.Drivers
         set Wins = Wins + 1,
             Reason = “Add 1999 Champion”
         where Name = ‘Kenny Brack’;
   quit;
```

In the last two examples, update operations to the data file result in the addition of two observations to the audit trail, a before image and an after image. We can verify the last three updates by printing a more complete version of the audit trail. To view the contents of the audit trail, submit the following code:

```
   proc print data=Indy.Drivers (type=audit
      keep=Name Reason Starts Wins _atopcode_);
   run;
```

The last six observations in the audit trail now contain the following data:

Name	Starts	Wins	Reason	_ATOPCODE_
Steve Knapp	1	0		DR
Steve Knapp	2	0	Add 1999 Start	DW
Tyce Carlson	1	0		DR
Tyce Carlson	2	0	Add 1999 Start	DW
Kenny Brack	3	0		DR
Kenny Brack	3	1	Add 1999 Champion	DW

The ‘Starts’ column was updated for drivers Steve Knapp and Tyce Carlson, and the ‘Wins’ column was updated for driver Kenny Brack. The observations with the ‘DR’ value for variable ‘_ATOPCODE_’ contain the before images, and the records with the ‘DW’ value contain the after images.

We can now add the 1999 champion to the INDY.CHAMPIONS data file. To add the new data along with a reason code, submit the following code:

```
   proc sql;
      insert into Indy.Champions
         values(‘Kenny Brack’, 1999,
                ‘Add 1999 Champion’);
   quit;
```

Adding the new record to the INDY.CHAMPIONS data file also results in an observation being added to its audit trail. To view the contents of the audit trail, submit the following code:

```
   proc print data=Indy.Champions (type=audit
      keep=Name Reason Year _atopcode_;
   run;
```

A single observation now exists in the INDY.CHAMPIONS audit trail.

Name	Year	Reason	_ATOPCODE_
Kenny Brack	1999	Add 1999 Champion	DA

The INDY.DRIVERS and INDY.CHAMPIONS data files now accurately contain the names of all drivers who have driven in the race and those who have won the race, from its beginning in 1911 through 1999. The same steps used to process the 1999 data can be used again in 2000.

CONCLUSION

The integrity constraints and audit trail features are new to Version 7 and Version 8 of the SAS System. The integrity constraints feature can be used to define and enforce data validation rules as part of the SAS data file, thus reducing the amount of data validation code that needs to be written into your applications. The audit trail feature can be used to closely monitor the update activities being performed on your data files, and gives you the ability to alter or reconstruct these updates after the fact. Together, these features can be used to better control and monitor updates to your SAS data files, while at the same time making you more productive.

REFERENCES

SAS Institute Inc. (1999), *SAS Language Reference: Concepts, Version 8*, Cary, NC: SAS Institute Inc.

ACKNOWLEDGMENTS

Example data for this paper were obtained from the official website of the Indy Racing League http://www.indyracingleague.com and the official website of the Indianapolis 500 http://www.indy500.com.

CONTACT INFORMATION

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

Gary S. Franklin
SAS Institute Inc.
11920 Wilson Parke Avenue
Austin, TX 78726
Phone: (512) 258-5171
Fax: (512) 258-3906
Email: Gary.Franklin@sas.com

Art Jensen
SAS Institute Inc.
Suite 1950
6400 S. Fiddler’s Green Circle
Englewood, CO 80111
Phone: (303) 290-9112
Fax: (303) 290-9195
Email: Art.Jensen@sas.com