„Integrity Constraints“ - Plausibilitätsüberprüfung in SAS Datensätzen

Sven Wichmann
HMS Analytical Software GmbH
Rohrbacher Straße 26
69115 Heidelberg
sven.wichmann@analytical-software.de

Zusammenfassung

Mit Integrity Constraints kann die Plausibilität und Qualität von Daten in SAS Datensätzen sichergestellt werden. Dazu können vorhandene Regeln aus Metadaten oder Anforderungsbeschreibungen für die Plausibilitätsüberprüfung eines Datensatzes herangezogen werden. Durch den Einsatz von Integrity Constraints werden diese Regeln Teil der Metadaten des Datensatzes und sichern so die Integrität der Daten.
Dieser Beitrag gibt einen Überblick über die verschiedenen Typen von Integrity Constraints und stellt beispielhaft dar, wie diese angelegt und verwendet werden können. Zudem werden die Vor- und Nachteile der Integrity Constraints aufgezeigt.

Schlüsselwörter: SAS Base, Datenvalidierung, Datenkonsistenz

1 Einleitung

Integrity Constraints (IC’s) sind ein Satz von Datenvalidierungsregeln, die spezifiziert werden können um die Inhalte, die in Datensätzen gespeichert werden können zu beschränken. Dadurch kann die Validität und Plausibilität eines Datensatz sichergestellt werden. Es soll gezeigt werden wie einige dieser Validierungsregeln erstellt und angewendet werden. Auch auf die Vor- und Nachteile der Validierungsregeln soll eingegangen werden.
Die gezeigten Beispiele lehnen sich an ein gängiges Szenario aus der klinischen Forschung an, bei dem die Variableninhalte des SAS Datensatzes mit den Definitionen eines Begleitdokuments, dem define.xml, übereinstimmen müssen. Dies bezieht nicht nur die Attribute der Variablen mit ein sondern auch deren mögliche Inhalte.

2 Einführung in Integrity Constraints

Es gibt zwei Typen von Integrity Constraints:

1. Allgemeine Einschränkungen
2. Referenzielle Einschränkungen

Referenzielle Einschränkungen dienen dazu die Schlüsselvariablen zweier Datensätze miteinander zu verknüpfen. Damit kann sichergestellt werden dass diese Schlüssel über

Datensatzgrenzen hinaus konsistent bleiben. Die Referenzielle Einschränkungen sollen jedoch im Weiteren nicht vertieft werden.

Die allgemeinen Einschränkungen unterteilen sich in die vier Subtypen:

Tabelle 1: Integrity Constraints des generellen Typs

check	Beschränkt die Werte die in einer Variable gespeichert werden können auf einen Bereich von Werten oder eine Liste von spezifizierten Werten.
not null	Sorgt dafür dass die Variable Daten enthalten muss. „Null"-Werte bzw. „missings" sind nicht erlaubt.
unique	Die Werte die in der Variable gespeichert werden können dürfen sich nicht wiederholen. „missing"-Werte sind dabei erlaubt, dürfen aber auch nur einmalig vorkommen.
primary key	Ähnlich der „unique" Beschränkung, allerdings sind „missing"-Werte nicht zulässig. Zudem kann die „primary key" Eigenschaft nur auf eine Variable im Datensatz angewendet werden.

2.1 Erhalt der Integrity Constraints

Enthält ein Datensatz Integrity Constraints, so gelten diese Einschränkungen nur für diesen Datensatz. Eine Vererbung der Beschränkungen findet nicht statt. Wird zum Beispiel ein Datenschritt bei dem Eingangs- und Ausgangsdatensatz den gleichen Namen haben ausgeführt, so enthält der „Neue" Datensatz keinerlei Beschränkungen mehr. Gleiches gilt generell wenn ein neuer Datensatz aus einem Datensatz erstellt wird der Beschränkungen enthält. Integrity Constraints bleiben unter Verwendung folgender SAS Base Prozeduren und Bedingungen erhalten:

Tabelle 2: Prozeduren die Integrity Constraints bewahren

APPEND	Die Beschränkungen des mit `BASE=` spezifizierten Datensatzes bleiben erhalten. Beschränkungen die ggf. im mit `DATA=` spezifizierten Datensatz gehen verloren. Existiert der hinter `BASE=` spezifizierte Datensatz allerdings nicht, so bleiben die Beschränkungen des mit `DATA=` spezifizierten Datensatzes erhalten. APPEND erzeugt somit eine Kopie des mit `DATA=` spezifizierten Datensatzes.
COPY, CPORT & CIMPORT	Erhalten die Beschränkungen wenn der Parameter „CONSTRAINT=yes" angewendet wird. Andernfalls werden die Integrity Constraints entfernt.
SORT	Erhält die Integrity Constraints unter der Bedingung das kein `OUT=` Statement angegeben wird.
SQL	PROC SQL erhält die Beschränkungen wenn die Aktionen INSERT, UPDATE oder DELETE auf einen Datensatz angewendet werden der Integrity Constraints enthält.

Die folgenden zwei SAS Codeblöcke sollen zeigen wie Integrity Constraints verloren gehen bzw. erhalten bleiben. Der Datensatz mit dem Namen „dataset_with_IC" ist der Datensatz der die Beschränkungen enthält und der Datensatz „dataset_without_IC" ist ein kompatibler Datensatz ohne Integrity Constraints.

Integrity Constraints gehen verloren	Integrity Constraints bleiben erhalten

`data new_dataset;`
`  set dataset_with_IC`
`  dataset_without_IC;`
`run;`
	`proc append base=dataset_with_IC`
	`  data=dataset_without_IC;`
	`run;`

Inhaltlich und bezogen auf die Metadaten der Variablen sind die Ergebnisse beider Tabellenhälften identisch. Bedingt durch die Tatsache, dass der Datenschritt im linken Codebeispiel allerdings einen neuen Datensatz erzeugt, gehen die Integrity Constraints verloren. PROC APPEND dagegen erhält die Beschränkungen die der Datensatz „dataset_with_IC" enthält und fügt die Inhalte des hinter `DATA=` spezifizierten Datensatzes an.

3 Erzeugen von Integrity Constraints

Zum Erzeugen von Integrity Constraints können die drei SAS Prozeduren SQL, DATASETS und SCL verwendet werden.
Im Folgenden soll zunächst gezeigt werden wie Integrity Constraints beim Erstellen eines neuen Datensatzes mit PROC SQL erzeugt werden können.

```
proc sql noprint;
   create table adsl(label='Subject-Level Analysis Dataset')
    (
    USUBJID  char  (20)    label="Unique  Subject  Identifier"  for-
mat=$20.,
    SEX     char (1)   label="Sex" format=$1.,
    AGE     num  (3)   label="Age" format=3.,
    ...,

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

    constraint CON_AGE check(age >= 18 and age <= 120)
               message = "Age must be between 18 and 120",

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

Mit dem obigen Code wir der Datensatz ADSL erzeugt. Die Erstellung der einzelnen Variablen ist der besseren Lesbarkeit wegen nur durch ein „..." angedeutet. Die Integrity Constraints werden mit dem Befehl „constraint" angelegt. Danach folgt ein Name der nachfolgenden Regel. Dieser muss nicht mit dem Variablennamen übereinstimmen und darf maximal 32 Zeichen lang sein. Nach dem Regelname folgt Art der Beschränkung

wie sie in Tabelle 1 aufgeführt sind. Hinter der Festlegung des Integrity Constraints Typs folgt in Klammern die Festlegung der Beschränkung. Als Optionale Möglichkeit kann im Anschluss eine Meldung spezifiziert werden die dem Benutzer im SAS-Log erkenntlich macht warum eine Regel dazu führt das zum Beispiel ein PROC APPEND fehlschlägt. Um dem Benutzer einen Hinweis zu geben warum eine Aktion fehlschlägt, sollte von dieser Möglichkeit möglichst Gebrauch gemacht werden. Der Text für diese Meldung sollte nicht länger als 250 Zeichen lang sein.
Die folgende Tabelle gibt eine Übersicht über die oben erzeugte Beschränkung „CON_SEX“:

Tabelle 3: Details zur Definition einer Integrity Constraints

`CONSTRAINT`	Initialisiert die Erzeugung eines Integrity Constraints innerhalb des PROC SQL Aufrufes.
`CON_SEX`	Optionaler Parameter Frei zu definierender Name der folgenden Beschränkungsregel. Der Name darf keine Leerzeichen enthalten und maximal 32 Zeichen lang sein. Wird kein Name angegeben wird von SAS ein Name angelegt der für CHECK folgende Konvention hat „_CKxxxx_“. xxxx ist eine Fortlaufende Nummer beginnend bei 0001 die für jede weitere Beschränkung des gleichen Typs um eins hochgezählt wird.
`CHECK(sex in ('M','F'))`	Festlegung der Art der Beschränkung. Innerhalb der Klammern erfolgt die inhaltliche Spezifikation der Regel. Dies muss einer gültigen SAS `WHERE` Bedingung entsprechen.
`MESSAGE = "Valid values for variable SEX: 'M', 'F'"`	Optionaler Parameter Die Festlegung einer optionalen Meldung erleichtert im Fehlerfall das Auffinden derjenigen Inhalte die zum Scheitern der Datenintegration geführt haben.
`MSGTYPE=`	Nicht im obige Beispiel gezeigt. MSGTYPE kann in Verbindung mit MESSAGE dazu verwendet werden die Art der Benutzernachricht festzulegen. `MSGTYPE=NEWLINE` legt fest, dass die Benutzermeldung zusätzlich zur SAS System Fehlermeldung angegeben wird

Das folgende Beispiel wendet Integrity Constraints auf einen bestehenden Datensatz an und benutzt dafür die Prozedur DATASETS.

```
proc datasets nolist;
  modify adsl_from_source;
    ic create CON_SEX = check(where=(sex in ('M','F')))
      message = "Valid values for variable SEX are either 'M' or
'F'.";
```

```
    ic create CON_AGE = check(where=(age >= 18 and age <= 120))
      message = "An invalid AGE has been provided.";
quit;
```

Die Unterschiede zum Erzeugen von Integrity Constraints mit PROC DATASETS im Vergleich zum oberen Beispiel mit PROC SQL besteht im Wesentlichen darin das mit PROC DATASETS die Prüfung mit einem `WHERE=()` eingeschlossen werden muss. Zudem muss, wenn ein Name angegeben wird, dieser mit einem Gleichheitszeichen von der Spezifikation der Beschränkung abgetrennt werden.
Sollen die Integrity Constraints auf einen nicht leeren Datensatz angewendet werden, werden die Regelsätze beim Hinzufügen getestet. Enthält der Datensatz ungültige Werte für eine der angelegten Integrity Constraints schlägt die Modifikation fehl und der Datensatz bleibt unverändert.
Die Eigenschaften des Erzeugten Datensatzes zeigen im Reiter „Integrity“ eine Liste der oben Erzeugten Beschränkungen wie dies in der folgenden Abbildung dargestellt ist.

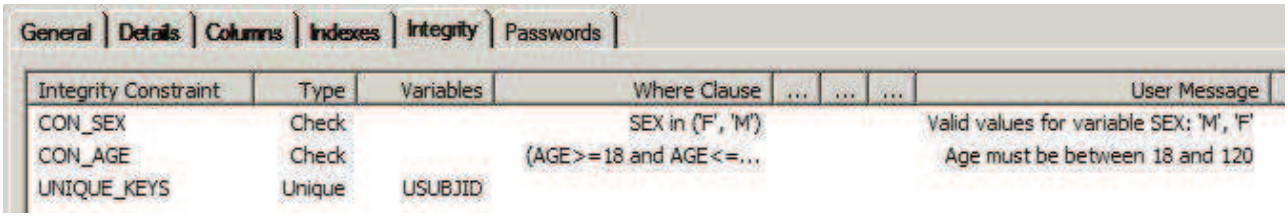

Abbildung 1: Eigenschaften der Datei ADSL nach Erzeugung von Integrity Constraints

4 Integrity Constraints im klinischen Anwendungsfall

Im klinischen Umfeld spielt die Konsistenz zwischen den Datensatz Metadaten und den an die Behörden übermittelten Daten ein wichtige Rolle. Die Metainformationen werden im Define.xml spezifiziert und mit an die Regulierungsbehörden übermittelt. Neben allgemeinen Metadaten wie Variablenlänge oder Variablenformat spielt auch die „controlled terminology“ eine wichtige Rolle. Auf diese Weise wird festgelegt welche Werte eine bestimmte Variable enthalten kann. Zudem legen die Implementierungsregeln auch fest, dass bestimmte Variablen nicht leer sein dürfen. Zusätzlich müssen auch die Schlüsselvariablen spezifiziert werden.
Die drei genannten Anforderungen lassen sich mit den Integrity Constraints CHECK, NOT NULL und UNIQUE auf zuverlässige Weise sicherstellen.
Werden regelmäßig Behördeneinreichungen vorbereitet, besteht in der Regel eine Struktur um aus den Datensatzspezifikationen für die Programmierung ein define.xml und Datensatztemplates zu erzeugen. Diese vorhandenen Systeme können so angepasst werden, dass auch die Integrity Constraints auf die finalen Datensätze angewendet werden. Auf diese Weise ist sichergestellt das nicht nur die üblichen Metadaten wie Formate, Längen und Variablentitel mit den Spezifikationen übereinstimmen, sondern auch Metadaten wie die Schlüsselvariablen. Zudem kann gleichzeitig eine inhaltliche Validie-

rung erfolgen die die Anforderungen an die „controlled terminology“ und nicht leeren Variablen sicherstellt.

5 Integrity Constraints und Daten zusammen führen

Wie in Kapitel 2.1 beschrieben, gibt es nur wenige Prozeduren die die Integrity Constraints bewahren. Das folgende Codebeispiel zeigt ein nicht unübliches Vorgehen beim Erzeugen eins Enddatensatzes unter Zuhilfenahme eines Beispieldatensatzes.

```
data ads.adsl;        /* Finaler ADSL Datensatz */
   set template_adsl /* Leerer ADSL Templatedatensatz mit IC's */
       data_adsl;    /* Fertige ADSL Daten */
run;
```

Der erzeugte Datensatz „ADSL“ entspricht bezüglich der Variablenmetadaten den Vorgaben. Es werden alle Variablenmetainformationen wie Label, Länge und Format aus dem Beispieldatensatz übernommen. Die inhaltliche Kontrolle der Daten die durch die erzeugten Integrity Constraints sichergestellt werden soll, findet jedoch nicht statt. Der neu erzeugte Datensatz „ADSL“ enthält keinerlei Beschränkungsregeln mehr und eine Überprüfung der vorhandenen Regeln hat nicht stattgefunden.

Die folgenden drei Beispiele sollen zeigen wie man die vorhanden Daten mit vorhandenen IC-Regeln zusammenbringen kann. Für die Beispiele wird der in Kapitel 3 erzeugt ADSL-Beispieldatensatz verwendet. Dieser enthält Beschränkungen für „AGE“ und „SEX“. Das Alter darf zwischen 18 und 120 (Jahren) liegen und „SEX“ darf nur die Werte „M“ und „F“ haben.
Als Datensatz wird der in Tabelle 4 spezifizierte verwendet. Um die verschiedenen Funktionsweisen der Methoden darzustellen, enthält der Datensatz insgesamt 4 Beobachtungen die den vorhandenen Regeln nicht entsprechen. Diese sind in der Tabelle dick hervorgehoben und Unterstrichen.

Tabelle 4: Beispieldatensatz „trialdata“

_N_	USUBJID	SEX	AGE
1	101	M	45
2	102	**T**	52
3	103	M	**17**
4	104	F	62
5	105	**T**	**16**

5.1 Zusammenführen mittels PROC APPEND

Wie in Kapitel 2.1 gezeigt, ist PROC APPEND eine mögliche Prozedur um die eigentlichen Daten mit einem Beispieldatensatz der IC's enthält zusammenzuführen. Unter der Voraussetzung das alle sonstigen Bedingungen für ein PROC APPEND erfüllt sind kann mit folgendem Code der „Studiendatensatz“ mit dem Beispieldatensatz verbunden werden.

```
proc append base=adsl
            data=trialdata;
run;
```

Soll der obige Datensatz „trialdata“ mittels PROC APPEND an den ADSL-Beispieldatensatz angehängt werden passieren zwei Dinge:

1. Der „neue“ ADSL Datensatz hat 2 Beobachtungen
2. Es gibt zwei Warnmeldungen im SAS-Log

Im neuen ADSL Datensatz sind nur 2 Beobachtungen enthalten. Die Beobachtungen 2, 3 und 5 wurden wegen Verletzung der Validierungsregeln abgelehnt und sind nicht mit in ADSL aufgenommen worden.
Das SAS-Log liefert folgende Details zur APPEND Prozedur:

```
WARNING: Age must be between 18 and 120 Add/Update failed for data
set WORK.ADSL because data value(s) do not comply with integrity
constraint CON_AGE, 1 observations rejected.

WARNING: Valid values for variable SEX: 'M', 'F' Add/Update failed
for data set WORK.ADSL because data value(s) do not comply with in-
tegrity constraint CON_SEX, 2 observations rejected.
```

Für jede “verletzte” Validierungsregel gibt SAS eine Warnung heraus. Zudem wird die Anzahl der „Verstöße“ mit angegeben. Allerdings verstößt im obigen Beispiel die Beobachtung 5 gegen zwei der Validierungsregeln (CON_AGE und CON_SEX) im SAS-Log steht jedoch, dass nur eine Beobachtung wegen der Validierungsregel „CON_AGE“ abgelehnt worden ist. Der Grund dafür ist, dass die Beobachtung 5 schon vorher abgelehnt worden ist. Die Validierungsregeln CON_AGE kommt nicht mehr zum tragen.
Zudem ist aus der Fehlermeldung nicht ersichtlich, bei welcher Beobachtung die Werte vorkommen die zur Ablehnung geführt haben.

5.2 Zusammenführen mittels PROC SQL

Als weitere Prozedur kann PROC SQL verwendet werden um die Daten in den Beispieldatensatz zu schreiben.

```
proc sql;
   insert into adsl
   select USUBJID, SEX, AGE from trialdata;
quit;
```

Wird PROC SQL dazu verwendet die Daten in das ADSL Template zu schreiben, passieren folgende zwei Dinge:

1. Der „neue“ ADSL Datensatz bleibt leer
2. Es wird ein „Error“ im SAS-Log ausgegeben

Im Vergleich zum obigen Beispiel mit PROC APPEND, verwirft PROC SQL alle 5 Beobachtungen aus dem Datensatz „trialdata“ um die Konsistenz des Template-ADSL-Datensatzes zu sichern.
Das SAS-Log gibt folgende Meldungen aus:

```
ERROR: Valid values for variable SEX: 'M', 'F' Add/Update failed for
data set WORK.ADSL because data value(s) do not comply with integri-
ty constraint CON_SEX.

NOTE: Deleting the successful inserts before error noted above to
restore table to a consistent state.
```

PROC SQL beendet das „insert into“ sobald die erste Beobachtung gegen eine Validierungsregel verstößt. Weitere Meldungen über mögliche zusätzliche Verletzungen werden nicht ausgegeben. Zudem Informiert eine „NOTE“ darüber, dass alle gegebenenfalls erfolgreich durchgeführten „Inserts“ wieder gelöscht worden sind und der Ursprungsdatensatz wieder in seinen Originalzustand versetzt worden ist.

5.3 Hinzufügen von IC’s mit PROC DATASETS

Als weitere Möglichkeit IC’s auf einen Datensatz anzuwenden ist, die Beschränkungen direkt mit PROC DATASETS auf den Datensatz mit den Daten anzuwenden.

```
proc datasets nolist;
  modify trialdata;
    ic create CON_SEX = check(where=(SEX in ('M','F')))
       message = "Valid values for variable SEX: 'M', 'F'";
    ic create CON_AGE = check(where=(age >= 18 and age <= 120))
       message = "Age must be between 18 and 120";
quit;
```

Mit dem obigen Code sollen die IC’s auf den bereits existierenden und nicht leeren Datensatz angewendet werden. Dabei können folgende zwei Dinge Beobachtet werden:

1. Der Datensatz „trialdata“ verbleibt unverändert
2. Im SAS-Log wird eine Fehlermeldung ausgegeben

Da das Anwenden der „Integrity Constraints“ fehlschlägt, verbleibt der Ursprungsdatensatz unverändert. Den Validierungsregeln wiedersprechende Daten sind weiterhin im Datensatz vorhanden. Die im SAS-Log ausgegeben Fehlermeldung ist wie folgt:

```
ERROR: Integrity constraint CON_SEX was rejected because 2 observa-
tions failed the constraint.
```

Wie PROC SQL wird auch das Hinzufügen der IC’s zu einem existierenden Datensatz beim der ersten Verletzung der Validierungsregel mit einem Fehler beendet. Allerdings wird zusätzlich die Anzahl der Werte die zum Abbruch geführt haben mit ausgegeben. Eine Überprüfung der weiteren IC’s findet jedoch nicht statt. Zudem wird auch die benutzerdefinierte Fehlermeldung nicht mit ausgegeben.

6 Arbeiten mit IC‘s

Die Beispiele aus Kapitel 5 zeigen, dass bei der Arbeit mit IC’s einige Dinge beachtet werden müssen. Zum einen muss dafür Sorge getragen werden, dass die IC’s nicht verloren gehen und zum anderen ist das Erkennen derjenigen Beobachtungen die zum Abbruch der Aktionen führen mit den normalen Fehlermeldungen kaum möglich. Im besten Fall werden zumindest alle Beschränkungen aufgeführt die zum Ablehnen der Daten geführt haben.

6.1 Audit trail mit PROC SQL

Eine Möglichkeit detailliertere Informationen zu den abgelehnten Daten zu bekommen ist das Audit trail das für SAS Datensätze aktiviert werden kann. Das Audit trail kann mit PROC DATASETS aktiviert werden. Es ist ein spezieller Datensatz der Modifikationen an einem Datensatz festhält.

```
proc datasets library=work nolist;
   audit adsl;
   initiate;
quit;

%macro insert_obs(nobs=);
   proc sql;
   %do i = 1 %to &nobs.;
      insert into adsl
      select USUBJID, SEX, AGE from trialdata(firstobs=&i. obs=&i.);
   %end;
   quit;
%mend insert_obs;
%insert_obs(nobs=&_obs.);
```

Um für jede Beobachtung eine Überprüfung der IC’s durchzuführen muss allerdings jede Beobachtung einzeln zum Beispiel mit PROC SQL und INSERT INTO in den Datensatz mit den IC’s eingefügt werden.
Der oben gezeigte Code aktiviert zunächst das Audit trail für den Datensatz. Im folgenden Makro werden die Beobachtungen der Reihe nach vom Datensatz „trialdata“ in den Templatedatensatz ADSL geschrieben. Das Ergebnis ist mit dem Datensatz aus Kapitel 5.1 identisch. Auch hier enthält der aktualisierte ADSL Datensatz die zwei gültigen Inhalte. Im SAS-Log werden allerdings drei Fehlermeldungen ausgegeben. Der zuvor aktivierte Audit trail Datensatz enthält alle 5 Beobachtungen mit Details dazu, warum einige Beobachtungen nicht in den ADSL Datensatz aufgenommen werden konnten.

Tabelle 5: Auszug des Audit trail Datensatz zu ADSL

_N_	USUBJID	SEX	AGE	_ATOBSNO_	_ATOPCODE_	_ATMESSAGE_
1	101	M	45	1	DA	
2	102	**T**	52	0	EA	ERROR: Add/Update failed for data set

						WORK.ADSL ...
3	103	M	**17**	0	EA	ERROR: Add/Update failed for data set WORK.ADSL ...
4	104	F	62	2	DA	
5	105	**T**	**16**	0	EA	ERROR: Add/Update failed for data set WORK.ADSL ...

Die Spalte "_ATMESSAGE_" wurde der besseren Lesbarkeit wegen gekürzt. Sie enthält die komplette Fehlermeldung die auch im SAS-Log ausgegeben wird. Die Spalte „_ATOPCODE_" zeigt über den Code „EA" die fehlgeschlagenen Einfügungen an. Über den Audit trail Datensatz kann nun detaillierter Überprüft werden welche Beobachtungen abgelehnt worden sind und warum. Allerdings besteht auch hier das Problem, dass die Beobachtung Nummer 5 gegen zwei der IC's verstößt. Im Audit trail wird jedoch nur ein Verstoß dargestellt. Zudem wird ein unvollständiger Datensatz ADSL erzeugt.

6.2 Audit trail mit PROC APPEND

Die in 6.1 vorgestellte Methode erfordert, dass jede Beobachtung einzeln zum Zieldatensatz hinzugefügt werden muss. Dies verbraucht jedoch deutlich mehr Systemressourcen und ist zeitaufwändiger. Der Audit trail Datensatz kann jedoch auch mit PROC APPEND benutzt werden.

```
proc datasets library=work nolist;
   audit adsl;
   initiate;
quit;

proc append base=adsl
            data=trialdata;
run;
```

Der SAS Aufruf, die Meldungen im SAS-Log sowie der resultierende ADSL Datensatz ist identisch mit dem aus Kapitel 5.1 dargestellten. Der Audit trail Datensatz enthält nach dem PROC APPEND acht Beobachtungen. Zu den fünf Beobachtungen aus dem Datensatz "trialdata" kommen drei Beobachtungen die mit denen aus Tabelle 5 identisch sind. Bedingt durch die Tatsache das mit PROC APPEND nicht jede Beobachtung einzeln in den Datensatz geschrieben werden muss, ist diese Vorgehensweise deutlich performanter als das in Kapitel 6.1 gezeigte Beispiel. Allerdings bleiben auch hier die Einschränkungen bestehen, dass der Resultierende Datensatz ADSL unvollständig ist und Einträge die gegen mehrere IC's verstoßen nur einmal im Audit trail aufgeführt werden.

6.3 IC's weiter verwenden

Alle oben genannten Methoden helfen nur unzureichend bei der Suche nach der Ursache für eine Verletzung der Validierungsregeln. Der Audit trail Datensatz bietet schon deutlich mehr Informationen zu den fehlgeschlagenen Beobachtungen, enthält allerdings nicht alle Details wenn eine Beobachtung gegen mehr als eine IC verstößt.
Für die bei der Datensatzerstellung beteiligten ist es jedoch wichtig zu wissen welche Werte bei welcher Beobachtung dazu geführt haben das die Validierungsregeln nicht eingehalten wurden.
Sind die Validierungsregeln erstellt, kann mit einem PROC CONTENTS der vollständige Regelsatz in einen Datensatz geschrieben werden.

```
proc contents data=adsl out2=icrules;
run;
```

Der mit dem Outputparameter OUT2 spezifizierte Datensatz enthält eine Übersicht aller vorhanden IC's im angegeben Datensatz. Ein Auszug ist im folgenden Bildschirmauszug zu sehen.

NAME	TYPE	RECREATE	MESSAGE	WHERE
CON_AGE	Check	ic create CON_AGE = Check (where=((AGE>=18 and AGE<=120)));	Age must be between 18 and 120	(AGE>=18 and AGE<=120)
CON_SEX	Check	ic create CON_SEX = Check (where=(SEX in ('F', 'M')));	Valid values for variable SEX: 'M', 'F'	SEX in ('F', 'M')
UNIQUE_KEYS	Unique	ic create UNIQUE_KEYS = Unique (USUBJID);		
USUBJID	Index	Index create USUBJID / Unique ;		

Abbildung 2: Datensatz mit IC Regeln erstellt durch PROC CONTENTS

Die Inhalte dieses Datensatzes können verwendet werden um die Validierungsregeln einfach wiederherzustellen. Der Inhalt der Spalte „RECREATE" kann direkt in einem PROC DATASETS verwendet werden. Zudem können die Bedingungen in der Spalte „WHERE" in ein Programm eingebaut werden das die Regeln prüft und Verletzungen in einen separaten Datensatz schreibt. Die Variablen die dem Typ „Unique" zugeordnet sind, sind im Datensatz ebenfalls enthalten.

7 Fazit

Integrity Constraints bieten eine sehr sichere Methode die Validität eines Datensatzes sicherzustellen. Dabei bieten sie ein hohes Grad an Flexibilität, so dass auch sehr komplexe Bedingungen erstellt werden können. Allerdings sind die Rückmeldungen im Fall einer Verletzung der Validierungsregeln zu allgemein. Eine detaillierte Information bei welcher Beobachtung welcher Wert zum Ablehnen der Beobachtung geführt hat gibt es nicht. Auch die gezeigten Möglichkeiten mit einem Audit trail helfen nur bedingt. Erst in Kombination mit separaten Prüfprogrammen lassen sich etwaige Fehler in den Daten schnell finden.
Werden die IC Regeln direkt aus einem define.xml heraus erzeugt und darauf aufbauend die zur Fehlersuche verwendeten Prüfprogramme, ergibt sich ein System das alle Anforderungen an Konsistenz, Zuverlässigkeit und Benutzerfreundlichkeit eines Systems zur Datensatzerzeugung bietet.