Groß, aber noch nicht Big: Optimierung von Programmen für große Datensätze

Stefan Beimel
Merz Pharmaceuticals GmbH
Eckenheimer Landstr. 100
60318 Frankfurt am Main
stefan.beimel@merz.de

Zusammenfassung

Datensätze in klinischen Studien sind üblicherweise nicht so groß, dass besondere Aufmerksamkeit auf die Geschwindigkeit der Datenverarbeitung mit SAS gelegt werden muss. Anders sieht es aus, wenn Daten von vielen Studien in einer integrierten klinischen Datenbank im SDTM- oder ADaM-Format gespeichert werden. Hier können zumindest einzelne Datensätze deutlich größer als 1 Gigabyte werden. Das stellt oft die Geduld des SAS-Programmierers auf die Probe. Dieser Beitrag zeigt Techniken, wie das Arbeiten mit großen Datensätzen effizienter gestaltet werden kann. Insbesondere das Minimieren von Lese- und Schreiboperationen verkürzt die Programmlaufzeit oft erheblich.
Einige Techniken wurden oft beschrieben (z. B. WHERE Statement vs. IF Statement), einige sind relativ unbekannt (z.B. MODIFY Statement im Datenschritt). Aber auch der Datentransfer von und zu externen Datenbanken lässt sich innerhalb von SAS gut überwachen und optimieren.

Schlüsselwörter: große Datensätze, WHERE, IF, MODIFY, Performance, Tuning, Optimierung, I/O

1 Einführung

Es gibt viele Ansätze, um SAS Programme zu beschleunigen. Neben dem Kauf neuer Hardware ist sicher die Verringerung von Lese- und Schreiboperationen von und auf Festplatten die Maßnahme mit dem größten Einsparpotential. Darauf konzentriert sich auch dieser Artikel.
Es wird beschrieben, wie

- Datensätze möglichst klein gehalten werden können,
- in Programmen möglichst wenig gelesen und geschrieben werden kann,
- Wege des schnellsten Datentransfers genutzt werden können,
- die Performance überwacht werden kann.

2 Datensatzgröße

2.1 KEEP, DROP, WHERE und IF

Ein einfaches Mittel, um die Größe von Datensätzen zu verringern, ist es, nicht benötigte Beobachtungen und Variablen so früh wie möglich von der Bearbeitung auszuschließen.

WHERE und IF filtern dabei nicht benötigte Beobachtungen, wobei WHERE effizienter beim Einlesen von Daten ist (siehe 3.1). DROP und KEEP sind Anweisungen die nur benannte Variablen einlesen oder schreiben (KEEP) bzw. vom Einlesen bzw. Schreiben ausschließen (DROP). Insbesondere temporäre Variablen (z. B. Schleifenzähler), die im Datenschritt benötigt werden, sollten sofort gedroppt und nicht in den Ausgabedatensatz geschrieben werden.

KEEP, DROP und WHERE sind sowohl Datenschrittanweisungen als auch Datensatzoptionen.

2.2 LENGTH und COMPRESS

SAS legt Datensätze sehr speicherintensiv ab. Textvariablen werden immer in voller Länge gespeichert, egal ob Text in ihnen steht oder ob nur oder fast nur Leerzeichen enthalten sind.

Obs	
1	••…•
2	Hier•steht•ein•einsamer•Text••••••••••••••••••••…•
3	••…•
…	
5000	••…•

Abbildung 1: Beispieldatensatz mit 1 Textvariablen der Länge 200, • = Leerzeichen

Der Datensatz wie hier abgebildet hat eine Größe von 993 KB.

Das LENGTH Statement
Mit dem LENGTH Statement im Datenschritt kann die Länge einer Variablen festgelegt werden. Im Beispiel oben reichen 28 Zeichen aus, um alle relevanten Informationen abzubilden.

Die Größe des Datensatzes verringert sich erheblich von 993 KB auf 141 KB.

Obs	
1	• •
2	H i e r • s t e h t • e i n • e i n s a m e r • T e x t
3	• •
…	
5000	• •

Abbildung 2: Beispieldatensatz mit 1 Textvariablen der „optimalen“ Länge 28,
• = Leerzeichen

Vorsicht bei numerischen Variablen
Auch numerische Variablen haben eine Länge, die aber auf keinen Fall mit der Länge der sichtbaren, formatierten Zahl verwechselt werden darf! Erlaubt ist eine Länge von 3 bis 8 Byte. Acht ist die Voreinstellung. Die Länge hat Auswirkungen auf die Genauigkeit der Zahlen. Bei einer Länge von 3 Byte können nur ganze Zahlen bis 8192 (2^{13}) genau abgebildet werden. Dezimalzahlen werden immer ungenauer gespeichert, je kürzer die Länge einer Variablen ist.

Tipp
Die minimale Länge einer numerischen Variable ist 3 Byte, die einer Textvariablen 1 Byte, d. h. 1 Zeichen. Man kann also eine Variable, die nur die Werte 0 und 1 enthält, platzsparender in einer Textvariablen der Länge 1 unterbringen als in einer numerischen Variablen.

Die COMPRESS Option
Der Effekt der oben beschriebenen Methode hängt stark vom längsten, tatsächlich vorhandenen Inhalt einer Variablen ab. Eine zusätzliche Möglichkeit, SAS Datensätze platzsparend zu speichern, ist die COMPRESS Option. Dabei werden sich wiederholende Zeichen geeignet zusammengefasst.

Obs	
1	28x •
2	H i e r • s t e h t • e i n • e i n s a m e r • T e x t
3	28x •
…	
5000	28x •

Abbildung 3: Beispieldatensatz mit COMPRESS (illustrativ), • = Leerzeichen

Der Größe des Beispieldatensatzes verringert weiter von 993 KB auf 85 KB.
Die COMPRESS Option kann an verschiedenen Stellen des Programms gesetzt werden.

Datensatz-Option:

```
data neu(COMPRESS=YES);
  set alt;
run;
```

Libname-Option (alle neu erzeugten Datensätze in diesem Verzeichnis werden komprimiert):

```
libname … 'c:\...' COMPRESS=YES;
```

System-Option (alle neu erzeugten Datensätze werden komprimiert):

```
options COMPRESS=YES;
```

Bemerkungen

- CPU Zeitbedarf wird erhöht.
- In seltenen Fällen, wenn der Datensatz sowieso schon recht klein ist, kann die Dateigröße steigen.

3 I/O Operationen

3.1 WHERE und IF beim Einlesen

WHERE verhindert, dass komplette Beobachtungen eingelesen werden. Das kann entweder als Datensatz-Option beim Eingabedatensatz oder als WHERE Statement im Datenschritt erfolgen. Die Bewertung der WHERE Bedingung erfolgt vor dem Einlesen der Beobachtung in den Program Data Vector (PDV), die Bewertung einer IF Bedingung danach.

Datenschritt mit IF	Besser mit WHERE

`data dsout;`
`  set dsin;`
`  if` *`Bedingung`*`;`
`run;`
	`data dsout;`
	`  set dsin(where=(`*`Bedingung`*`));`
	`run;`
	
	oder
	
	`data dsout;`
	`  set dsin;`
	`  where` *`Bedingung`*`;`
	`run;`

Da für die Bewertung der Bedingung aber trotzdem Variablenwerte gelesen und verarbeitet werden müssen, ist der Effekt in Beispieldatensätzen moderat. Selbst wenn nichts gelesen wurde, d.h. die Bedingung immer falsch war, sparte WHERE nur etwa 10% Laufzeit für das reine Einlesen ein.

Tipp
Die Bedingung '0' wird schon in der Kompilierungsphase von SAS als falsch erkannt und mit der Note `WHERE 0 /* an obviously FALSE WHERE clause */` quittiert. Das kann z. B. genutzt werden, um leere Datensatzstrukturen aus vorhandenen Datensätzen zu erzeugen. Hier wird zur Laufzeit kein Vergleich durchgeführt und auch nichts eingelesen, also praktisch keine Zeit verbraucht.

```
data dsout;
  set dsin;
  where 0;
run;
```

Der Datenschritt funktioniert nicht mehr so effizient – allerdings mit dem gleichen Ergebnis – , wenn `IF 0` oder `WHERE 0=1` anstelle von `WHERE 0` verwendet wird. Diese Bedingungen werden nicht als generell falsch erkannt.

Ein großer Vorteil des WHERE ist, dass es als Datensatzoption verwendet werden kann und – zumindest in vielen Prozeduren – als eigene Anweisung. Damit ist Filtern eines Datensatzes direkt in einer Prozedur ohne vorherigen Datenschritt möglich.

```
proc sort data=dsin(where=(Bedingung))
          out=dsout;
  by var;
  where Bedingung;
run;
```

Um den optimalen Effekt zu erzielen, ist es wichtig, WHERE bereits in der DATA= Option zu verwenden (beim Einlesen) und nicht erst bei DSOUT= (beim Schreiben).

Bemerkungen:

- Da WHERE vor der Verarbeitung im Datenschritt angewandt wird, müssen alle angesprochenen Variablen bereits im Eingabedatensatz vorhanden sein. Es können keine berechneten Variablen verwandt werden. Allerdings ist möglich, Formeln und Funktionen in der Bedingung zu verwenden.
- Die Schleifenzählvariable _n_ hat bei WHERE und IF unterschiedliche Werte.

3.2 PROC APPEND

Wenn zwei Datensätze mit gleicher Struktur wie bei einem SET verbunden werden müssen, kann alternativ PROC APPEND benutzt werden. Der erste Datensatz wird dabei nicht gelesen, sondern nur der zweite Datensatz ans Ende des ersten 'angehängt'.

```
proc append base=gross new=klein;
run;
```

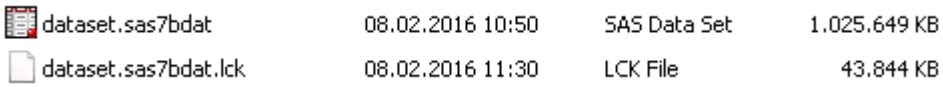
Im Beispiel wird aus allen Datensätzen in einem Verzeichnis die Variable USUBJID eingesammelt. Der Datensatz SUBJECTS_APPEND wächst dabei kontinuierlich.

```
*** alle Datensatznamen den Makrovariablen DS1 bis DS99
    zuweisen ***;
proc sql noprint;
   select memname into :ds1 - :ds99
```

```
   from dictionary.columns
   where libname="WORK" and name="USUBJID";

%let nds=&sqlobs;


*** Den kombinierten Datensatz für wiederholbare
    Ausführung löschen ****;
proc datasets nolist;
   delete  subjects_append;
run;


%macro append();
   %do i= 1 %to &nds;
      proc append base = subjects_append
                  new  = &&ds&i (keep=usubjid);
      run;
   %end;
%mend;
%append;
```

3.3 MODIFY statt SET

Wenn mit dem SET Statement ein Datensatz erzeugt wird, wird zunächst eine Datei namens *.sas7bdat.lck angelegt und in *.sas7bdat umbenannt, wenn der Datenschritt erfolgreich beendet wurde.
Haben Eingabe- und Ausgabedatei den gleichen Namen, kann man eine 'Kopie' des Datensatzes auf Betriebssystemebene 'wachsen' sehen:



Schneller als mit einem SET geht es in diesem Fall mit dem MODIFY Statement. Hierbei wird der Datensatz 'in place' geändert und dadurch I/O-Zeit gespart. Im Log wird angezeigt, wie viele Beobachtungen `rewritten`, `added` oder `deleted` wurden.

```
data dataset;
  modify dataset;
run;
```

Beispiel:
Es sollen alle numerischen Variablen eines Datensatzes gerundet werden.

Mit dem SET Statement könnte der Datenschritt so aussehen.

```
data lb;
  set lb;
  array _num {*} _numeric_;
  do i= 1 to dim(_num);
      _num[i] = round(_num[i] ,1E-12);
  end;
```

```
  drop i;
run;
```

Etwas komplizierter, dafür schneller in der Laufzeit, ist der entsprechende Datenschritt mit MODIFY. Hier werden nur die geänderten Beobachtungen geschrieben.

```
data lb;
  modify lb;
  array _num {*} _numeric_;
  changed=0;
  do i= 1 to dim(_num);
    temp=round(_num[i] ,1E-12);
    if _num[i] ne temp then do;
         _num[i] = temp;
         changed=1;
    end;
  end;
  if changed then replace;
run;
```

Ein paar Bemerkungen:

- Da der Ausgabedatensatz mit SET ein neuer Datensatz ist, werden einige Attribute nicht übernommen, z. B. SortedBy-Information und das Datensatzlabel. Mit MODIFY bleiben diese Informationen erhalten.
- Mit dem MODIFY Statement können keine neuen Variablen an den Datensatz angefügt werden – auch nicht 'aus Versehen' Hilfsvariablen wie `i`, `changed` und `temp` wie im Beispiel. Textvariablen können nicht verlängert oder verkürzt werden.
- Die Syntax im Datenschritt ist mit einem MODIFY etwas anders als beim SET Statement [1]. Um eine Beobachtung zu verändern, muss sie mit REPLACE ersetzt werden, wobei nur ein REPLACE pro Beobachtung zulässig ist.

3.4 DATA _NULL_

Wenn mit einem Datenschritt kein Datensatz erzeugt werden soll, kann das verhindert werden, indem der 'Ausgabedatensatz' _NULL_ genannt wird.

```
data _null_;
  SAS Statements;
run;
```

Das kann z. B. der Fall sein, wenn

- mit CALL SYMPUT Makrovariablen gefüllt werden sollen,
- mit PUT in eine externe Datei geschrieben werden soll,
- mit PUT Informationen ins Log geschrieben werden sollen oder
- mit CALL EXECUTE SAS Code erzeugt werden soll.

3.5 Merge mit Formaten

Eine Möglichkeit, Sortierungen und damit Lese- und Schreiboperationen zu sparen, ist die Verwendung von Formaten zum Mergen von Dateien. Hierbei werden der Formatname, die Startwerte (Codes) und Label (Decodes) zuerst in einen Datensatz geschrieben und dann mit der CNTLIN= Option des PROC FORMAT in ein Format umgewandelt. Die Rolle der Schlüsselvariablen (BY Variablen) übernehmen hier die Startwerte des Formats. Anders als beim MERGE Statement ist es möglich, mit verschiedenen Formaten auch verschiedene Schlüssel in einem Datenschritt zu nutzen (mehrere BY Statements).

Konventionelles PROC FORMAT (zur Illustration der Bedeutung der Variablen FMTNAME, START und LABEL im CNTLIN Datensatz):

```
proc format;
  value FMTNAME
     START1 = LABEL1
     START2 = LABEL2
        . . .
     other  = ' ';
run;
```

Beispiel, um die Variable SEX (Datensatz DM) an den Datensatz LB zu mergen:

```
** Lösung mit konventionellem Datenschritt (verkürzt) ***;
* data lbsex;
*   merge lb dm;
*   by usubjid;
* run;

** _fmtsex dient als Eingabe für proc format, um das Format
   $sex zu erzeugen;

data _fmtsex(keep=fmtname start label);
  set dm (keep=usubjid sex)
      end=lastobs;

  length fmtname $4;
  fmtname = "$sex";
  output;
  *** Falls USUBJIDs in LB, aber nicht in DM vorhanden sind;
  if lastobs then do;
    usubjid = "other";
    sex     = " ";
    output;
  end;
  rename usubjid=start
         sex=label;
run;
```

VIEWTABLE: Work._fmtsex

	FMTNAME	START	LABEL
1	$sex	Subj01	Female
2	$sex	Subj02	Female
3	$sex	Subj03	Male
4	$sex	other	

Der Datensatz _fmtsex dient als Eingabedatensatz (CNTLIN=) für das folgende PROC FORMAT.

```
proc format cntlin=_fmtsex;
run;

** Mit Hilfe des Formats $sex wird die Variable sex erzeugt;
data lbsex;
  set lb;
  length sex $1;
  sex=put(usubjid,$sex.);
run;
```

Mit einem konventionellen MERGE müssten beide Datensätze sortiert werden. Soll die ursprüngliche Sortierung wieder hergestellt werden, ist noch ein zusätzliches PROC SORT notwendig; ist die ursprüngliche Sortierung unbekannt, benötigt man sogar noch einen weiteren Datenschritt.

Nachteile der Methode 'Merge mit Formaten':

- Der Code ist komplizierter als beim MERGE.
- Besteht der Schlüssel aus mehreren Variablen, müssen sie in der START-Variablen des Formats geeignet zusammengefasst werden.
- Die Variablenattribute (im Beispiel SEX) werden nicht übernommen.
- Sollen mehrere Variablen gemergt werden (z. B. neben SEX noch AGE und BMI), müssen verschiedene Formate verarbeitet werden.

3.6 Metadaten mit PROC DATASETS ändern

Um Metadaten zu ändern, ist es nicht nötig, den kompletten Datensatz in einem Datenschritt zu lesen und zu schreiben. Es ist deutlich schneller, mit PROC DATASETS nur die Beschreibung des Datensatzes bzw. der Variablen zu ändern.

Zu den Metadaten zählen u. a.

- Name des Datensatzes,
- Label des Datensatzes,
- Namen von Variablen,
- Formate und Informate von Variablen, und
- Label von Variablen.

3.7 Datenschritte und Prozeduren kombinieren

Falls der Ausgabedatensatz eines Datenschritts ausschließlich als Eingabedatensatz für einen nachfolgenden Datenschritt oder eine Prozedur dient, können diese Einzelschritte oft zusammengefasst werden.

Beispiel 1

Vorher	Zusammengefasst

`data ds1;`
`  set dsin;`
`  SAS Statements;`
`run;`
 	
`data dsout;`
`  set ds1;`
`  weitere SAS Statements;`
`run;`
	`data dsout;`
	`  set dsin;`
	`  SAS Statements;`
	`  weitere SAS Statements;`
	`run;`

Beispiel 2
Hier werden in einem Datenschritt ausschließlich Statements benutzt, die auch als Datensatzoption verwendet werden können.

Vorher	Zusammengefasst

`data ds1;`
`  set dsin;`
`  WHERE  ...;`
`  RENAME ...;`
`  KEEP   ...;`
`  DROP   ...;`
`run;`
 	
`proc ... data=ds1;`
`  ...`
`run;`
	`proc ... data=dsin`
	`  (WHERE  = (...)`
	`  RENAME = (...)`
	`  KEEP   = ...`
	`  DROP   = ...`
	`  );`
	`  ...`
	`run;`

3.8 STOP

Das STOP Statement beendet einen Datenschritt unmittelbar. Im folgenden Beispiel wird der Makrovariablen MVAR_NOBS die Anzahl der Beobachtungen im Datensatz DSIN zugewiesen. Ohne STOP würde der gesamte Datensatz gelesen werden: mit dem gleichen Ergebnis, nur länger.

```
data _null_;
  call symput("mvar_nobs",put(var_nobs,best.));
  stop;
  set dsin nobs=var_nobs;
run;
```

Da NOBS= bereits während der Kompilierungsphase zugewiesen wird, muss das SET Statement noch nicht einmal ausgeführt werden – es steht nach dem STOP. Wichtig ist hier, dass CALL SYMPUT vor dem SET ausgeführt: Sollte der Datensatz keine Beobachtungen beinhalten, würde MVAR_NOBS sonst entweder nicht vorhanden sein oder einen alten Wert haben.

4 Datentransfer

Wenn SAS nicht ausschließlich auf einem lokalen Rechner verwendet wird, gibt es je nach Architektur des Netzwerkes sehr unterschiedliche Übertragungsgeschwindigkeiten zwischen den einzelnen Komponenten des Netzwerks.

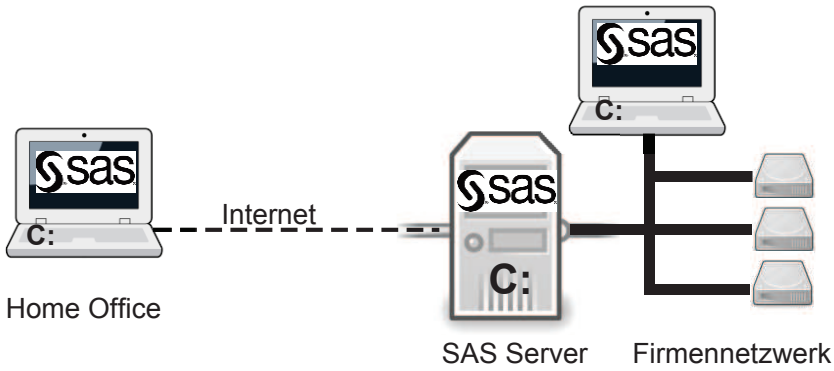


Abbildung 4: Beispielhafte, vereinfachte Architektur einer Netzwerkumgebung mit SAS Installationen

Eine grobe Abschätzung der Übertragungsgeschwindigkeiten kann man erhalten, wenn man die Zeit misst, die für das Kopieren von Dateien benötigt wird (siehe Tabelle 1). Daraus ergibt sich, dass die Festplatte des SAS Servers die mit Abstand schnellste Übertragungsrate hat, gefolgt von den Festplatten im Firmennetzwerk und der lokalen Festplatte des Laptop. Die Datenübertragung über das Internet ist die bei weitem langsamste Methode. Die lokale Festplatte des Servers (≈700 MB/s) ist so schnell, dass manche Optimierungen der I/O kaum ins Gewicht fallen.

Tabelle 1: Beispiel des Zeitbedarfs für das Kopieren einer Datei der Größe 1 Gigabyte

lokale Festplatte des Servers	<	lokale Festplatte des Laptops	≈	Laufwerk im Firmennetzwerk	≪	privates Internet *
1,8 s		12 s		13 s		65 Minuten

* Über den lokalen Explorer des Laptops, wenn er über das Internet im Firmennetzwerk angemeldet ist. Der Kopiervorgang einer Datei bedeutet ein Download und ein Upload über das Internet. Das Ergebnis hängt stark von der Geschwindigkeit des persönlichen Anbieters ab.

Daraus lassen sich folgende Empfehlungen ableiten:

- Das WORK Verzeichnis sollte auf einer lokalen Festplatte des Rechners liegen, auf dem SAS installiert ist.

- So lange wie möglich auf WORK oder einem anderen lokalen Verzeichnis arbeiten und erst ganz zum Schluss im Netzwerk speichern.
- Möglichst mit den SAS Server arbeiten und nicht mit einer SAS Installation auf einem Laptop.
- Möglichst nicht die lokale SAS Installation nutzen, wenn der Laptop über das Internet mit dem Firmennetz verbunden ist und auf SAS Datensätze im Firmennetz zugegriffen werden muss.

Generell gilt für das Arbeiten aus dem Home Office, d.h. wenn der Laptop über das Internet mit dem Firmennetzwerk verbunden ist: Wenn die Funktionalität, die man benötigt, auf einem Server innerhalb des Netzwerks vorhanden ist, sollte man diese von dort nutzen. Das gilt u.a. für das Kopieren, Verschieben und Zippen von Dateien (Windows Explorer), das Anschauen von SAS Dateien (SAS Viewer), Downloads und Uploads mit FTP, Nutzen der Internetverbindung der Firma, Textverarbeitung, …

5 Überwachung

5.1 (FULL)STIMER

Die SAS Optionen STIMER und FULLSTIMER geben Auskunft über Gesamtzeit eines Datenschrittes bzw. einer Prozedur, über die Speichernutzung und über die CPU Zeit.

5.2 SASTRACE

Mit der Option SASTRACE kann beobachtet werden, welche Befehle von SAS und welche von externen Datenbanken, z. B. Oracle, durchgeführt werden. Ziel ist es, dass möglichst viele Befehle direkt von der Datenbank selbst durchgeführt werden, damit kein unnötiger Datentransfer stattfindet.

Für das Beispiellog wurden folgende Optionen gesetzt:

```
options sastrace=',,,d' sastraceloc=saslog nostsuffix;
```

Im Beispiel sollen aus einer Oracletabelle nur diejenigen Beobachtungen selektiert werden, die einen entsprechenden Eintrag im SAS Datensatz SASdsin haben.

```
proc sql;
  create table dsout as
  select  oracle.value_text
         ,oracle.proc_variable_name
         ,oracle.response_id
         ,oracle.response_entry_ts
         ,oracle.discrepancy_entry_id
  from  SASdsin
           join
        rxc.validation_reported_valuest  oracle
  on    SASdsin.discrepancy_entry_id = oracle.discrepancy_entry_id
  where proc_variable_name like '%.%' or proc_variable_name is null;
quit;
```

Auszug aus dem Log:

```
ODBC_34: Prepared: on connection 1
SELECT   "DISCREPANCY_ENTRY_ID", "VALUE_TEXT", "PROC_VARIABLE_NAME",
"RESPONSE_ID", "RESPONSE_ENTRY_TS"  FROM
rxc.VALIDATION_REPORTED_VALUEST   WHERE   (  (  "PROC_VARIABLE_NAME"
LIKE  '%.%' ) OR  ( "PROC_VARIABLE_NAME" IS  NULL  )  )

NOTE: PROCEDURE SQL used (Total process time):
      real time           51.65 seconds
      user cpu time       26.80 seconds
```

Bemerkenswert ist, dass die ON-Bedingung des JOIN nicht zur Datenbank übertragen wurde. Es wurden also erst einmal alle Beobachtungen selektiert und der JOIN danach von SAS durchgeführt. Die Abfrage konnte verbessert werden, indem die JOIN Bedingung in die WHERE Bedingung integriert wurde und damit auf den Vergleich mit dem SAS-Datensatz verzichtet werden konnte.

Optimierte Version:

```
proc sql noprint;
select distinct discrepancy_entry_id
        into :discrepancy_entry_ids separated by ' '
 from SASdsin;
create table dsout as
 select    value_text
          ,proc_variable_name
          ,response_id
          ,response_entry_ts
          ,discrepancy_entry_id
 from  rxc.validation_reported_valuest
 where (proc_variable_name like '%.%' or proc_variable_name is null)
          and
       discrepancy_entry_id in(&discrepancy_entry_ids);
quit;
```

Auszug aus dem Log:

```
ODBC_27: Prepared: on connection 1
SELECT       "VALUE_TEXT",    "PROC_VARIABLE_NAME",    "RESPONSE_ID",
"RESPONSE_ENTRY_TS", "DISCREPANCY_ENTRY_ID"  FROM
rxc.VALIDATION_REPORTED_VALUEST   WHERE   (  (  (  "PROC_VARIABLE_NAME"
LIKE  '%.%' ) OR  ( "PROC_VARIABLE_NAME" IS  NULL  )  ) AND  ( (
"DISCREPANCY_ENTRY_ID"  is  NULL   or
"DISCREPANCY_ENTRY_ID" = 242413601  or
"DISCREPANCY_ENTRY_ID" = 242413701  or
...

NOTE: PROCEDURE SQL used (Total process time):
      real time           0.06 seconds
```

Die Abfrage benötigt nun statt ca. 50 Sekunden nur noch 0,06 Sekunden!

5.3 %TIME

Das Makro %time kann genutzt werden, um die Zeit zwischen Programmschritten zu messen. Bei wiederholten Aufrufen von %time gibt es Auskunft über die verstrichene Zeit seit dem jeweils letzten Aufruf von %time.

```
%macro time(msg);
   %global lasttime;
   %if &lasttime = %then %let lasttime= %sysfunc(time());
   %put INFO: &msg
      %sysfunc(round(%sysevalf(%sysfunc(time())-&lasttime),0.01)) s;
   %let lasttime= %sysfunc(time());
%mend;
```

6 Weiterführendes / Fazit

Neben den oben beschriebenen Maßnahmen gibt es noch eine Vielzahl anderer Möglichkeiten, die Laufzeit eines SAS Programms zu verkürzen:

- Hash Objekte
- Indizes
- Views
- Threads
- Datensatzoption OBS= beim Debuggen
- …

Es gibt viele Möglichkeiten, die Performance eines SAS Programmes zu erhöhen. Welche tatsächlich zum Erfolg führt, ist sehr individuell und muss ausprobiert werden. Bei Makros, die häufig verwendet werden – auch wenn die verarbeiteten Datensätze relativ klein sind –, ist es sinnvoll, mehr Aufwand in die Performancemaßnahmen zu investieren als bei Programmen, die sehr selten laufen.

Literatur

[1] SAS Online Hilfe, http://support.sas.com/documentation/94/index.html