Tipps und Tricks für den leichteren Umgang mit der SAS Software

Carina Ortseifen
Universitätsrechenzentrum Heidelberg
Im Neuenheimer Feld 293
69120 Heidelberg
carina.ortseifen@urz.uni-heidelberg.de

Grischa Pfister
iCASUS GmbH
Vangerowstraße 2
69115 Heidelberg
g.pfister@icasus.de

Heribert Ramroth
Universitätsklinikum Heidelberg
Institut für Public Health
Im Neuenheimer Feld 324
69120 Heidelberg
heribert.ramroth@uni-heidelberg.de

Marianne Weires

Deutsches Krebsforschungszentrum
Im Neuenheimer Feld 580
69120 Heidelberg
m.weires@dkfz.de

Zusammenfassung

In Form von kurzen Beiträgen werden nützliche Lösungen zu Problemen und Fragestellungen vorgestellt, die bei der täglichen Arbeit mit der SAS Software auftreten können. Es werden dabei nicht unbedingt neue Prozeduren, Optionen oder Module vorgestellt. Stattdessen soll die effektive Anwendung vorhandener Anweisungen und Prozeduren an Beispielen aufgezeigt werden.

1. Wohin mit den Pfadangaben? – Verwendung eines Vorlaufprogramms
2. PROC SQL - Auffinden von unterschiedlichen Tabelleneinträgen
3. Der DATA Step Debugger
4. Parallelisierung mit MP Connect
5. Automatisches Ermitteln der Achsenbeschriftung
6. PROC FCMP – Eigene Funktionen im Datenschritt erstellen
7. Endlich richtig sortieren mit der Option Sortseq der Prozedur SORT
8. SAS Batch – Checkpoint und Restart-Modus
9. Änderungen in SAS/GRAPH

Schlüsselwörter: %MACRO-Anweisung, _METHOD, _TREE, Array, Autoexec.sas, BasicProjectSettings.sas, COLLATION=, Datenschritt, DISTINCT Subroutine, FCMP-Prozedur, Funktion, Globale Makrovariable, LINGUISTIC=, MIDPOINTS=, NUMERIC_COLLATION=Option SORTSEQ=, Paket, Pfadangaben, PHONEBOOK, PROC FCMP, PROC SORT, PROC SQL, ODS RTF Rekursion, SAS/Graph, SORT-Prozedur, SORTSEQ=, SQL DISTINCT Subroutine

1 Wohin mit den Pfadangaben? Verwendung eines Vorlaufprogramms

(Heribert Ramroth)

Pfadangaben in SAS Programmen - Ausgangssituation

Stellen Sie sich vor,

- Sie arbeiten sowohl auf dem Institutsserver als auch unterwegs auf einem Laptop
- Ihr Datenbankmanager ändert mal wieder die Ordnerstruktur auf dem Server
- Sie erhalten Programme & Daten, die auf Ihrem Rechner erneut ablaufen sollen

Bei Änderungen in den Ordnerstrukturen sind in allen SAS Programmen mit Pfadangaben Änderungen erforderlich. Dies ist

- zeitaufwendig
- ärgerlich
- vermeidbar

Im Folgenden wird eine einfache Möglichkeit gezeigt, dies zu vermeiden:

- Sie nehmen alle Pfadangaben aus den Programmen heraus und definieren *Libnames* und *Filenames* in einem Vorlaufprogramm, z.B.
  - Autoexec.sas
  - BasicProjectSettings.sas
- Noch besser: Sie übergeben den Pfad an eine Makro-Variable

Vorteil:

- Bei Änderungen muss nur dieses aktualisierbare Vorlaufprogramm verändert werden. Im einfachsten Fall nur die Makro-Variable.

Die Definition von Libnames und Filenames in einem Vorlaufprogramm erhöht in jedem Fall die Übersicht. Im Falle der Verwendung einer globalen Makrovariablen, auf die auch später beliebig in den Programmen zurückgegriffen werden kann, können Libnames und Filenames ohne explizite Pfadnennung auch später definiert werden.

Beispiel einer Ordnerstruktur

Default Directory (main path)

C:\MyFolder\MyProject\

Sub directories, e.g.

C:\MyFolder\MyProject\OriginalData
C:\MyFolder\MyProject\MyData
C:\MyFolder\MyProject\Programs
C:\MyFolder\MyProject\Results
C:\Irgendwo\MyMacros

Eine Vorlaufdatei, die grundlegende Optionen für die gerade eröffnete SAS Sitzung festlegt, könnte wie folgt aussehen. Manchen Anwendern ist solch eine Datei möglicherweise unter dem Namen „Autoexec.sas“ bekannt. Ich möchte die Datei aufgrund der Zielsetzung hier jedoch einfacher „BasicProjectSettings.sas“ nennen.

```
options pagesize=54 pageno=1 nocenter ;
libname a "C:\Myfolder\MyProject\OriginalData";
libname b "C:\Myfolder\MyProject\MyData";

* Bereitstellung möglicherweise verwendeter SAS-Makros;
%include "C:\Myfolder\MyProject\Programs\unistats.sas";

* RTF-Output (“Word”-Documents);
filename rtfdummy "C:\MyFolder\MyProject\Results\Dummy.rtf";
```

Die Verwendung einer festen Ausgabedatei (hier Dummy.rtf) in einem festgelegten Pfad soll hier nochmals gesonderte Beachtung finden:
Im Allgemeinen ist die RTF-Ausgabe einer Datei nicht endgültig und erfordert Ergänzungen oder verschönernde Nachbearbeitungen. Wird die Ausgabe in eine Aufgabenspezifische RTF-Datei geleitet, dann wird sie bei einem späteren – möglicherweise unbeabsichtigten – Programmdurchlauf überschrieben.
Daher ist es – meines Erachtens – am geschicktesten, die RTF-Ausgabe immer in eine temporäre Dummy-Datei auszugeben, diese in einem manuellen Schritt zu kontrollieren, ergänzen und verschönern. Abschließend wird das überarbeitete Ergebnis dann in einer Ergebnisdatei gespeichert, die zu keinem Zeitpunkt Gefahr läuft programmtechnisch überschrieben zu werden.

Wie oben ersichtlich, gibt es offensichtlich bei einem wohlstrukturierten Projekt immer einige Wiederholungen von Teilen des Pfadnamens, d.h. Wiederholung von Text. Optimieren lässt sich die Verwendung von wiederholtem Text durch Einsatz einer oder mehrerer Makro Variablen.

```
* Definiere Makro-Variable “Root”;
%LET Root=C:\Myfolder\MyProject ;

libname a "&Root.\OriginalData";
libname b "&Root.\MyData";

%include "&Root.\Programs\unistats.sas";

* RTF-Output (“Word”-Documents);
filename rtfdummy "&Root.\Results\Dummy.rtf";
```

Der Hauptvorteil der Verwendung der Makro-Variablen *Root* ist offensichtlich: Einfache Anpassung an veränderte Datenstrukturen wird durch alleinige Änderung der Root-Variablen erreicht. Bei komplexeren Ordnerstrukturen können natürlich auch mehrere Makro-Variablen definiert werden.
Besonders soll hier noch auf die Definition und Verwendung einer weiteren Makro-Variablen hingewiesen werden.

Das *i*-Tüpfelchen: &OdsRtfOpen.

Man betrachte die folgenden sinnvollen ODS Optionen:

```
ods rtf file=rtfdummy bodytitle startpage=no style=styles.journal;
```

startpage=no	Lästige Seitenumbrüche werden vermieden. Insbesondere müssen diese nicht manuell später aus dem RTF-Dokument entfernen werden.
bodytitle	Titel werden direkt vor das Prozedurergebnis und nicht in die Kopfzeile geschrieben.
style=styles.journal	ein optisch netter Ausgabestil (Alternative: styles.minimal)

Diese ODS Optionen werden nun in die Makro-Variable OdsRtfOpen geschrieben:

```
%LET OdsRtfOpen = %STR(<ODS Aufruf plus Optionen>);
```

Aufruf später im Programm:

Statt:

```
ods rtf open <ODS OPTIONEN>;
    relevante Ausgabe;
ods rtf close;
```

einfacher:

```
&OdsRtfOpen.;
    relevante Ausgabe;
ods rtf close;
```

Zusammenfassung

```
options pagesize=54 pageno=1 nocenter ;
%let Root=C:\Myfolder\MyProject ;
libname a "&Root.\OriginalData";
libname b "&Root.\MyData";
%include "&Root\Programs\unistats.sas";
filename rtfdummy "&Root.\Results\Dummy.rtf"; *…csv… pdf… etc.;
%LET OdsRtfOpen =
     %STR(ods rtf file=rtfdummy bodytitle startpage=no
     style=styles.journal);
```

2 Auffinden unterschiedlicher Einträge mit PROC SQL

(Marianne Weires)

Ziel dieses Tipps ist es die Verwendung des Schlüsselwortes `DISTINCT` zur Auffindung unterschiedlicher Einträge im Rahmen von `PROC SQL` zu veranschaulichen. Zur weiteren Verarbeitung werden die unterschiedlichen Werte als Array von Makrovariablen oder als Makrovariablen-Array gespeichert. Zusätzlich werden die nicht dokumentierten Anweisungsoptionen `_METHOD` und `_TREE` in Bezug auf `DISTINCT` kurz vorgestellt.

SQL steht für Structured Query Language und ist eine weit verbreitete Datenbanksprache zur Definition, Abfrage und Bearbeitung von Daten in relationalen Datenbanken. PROC SQL ist die SAS Implementierung von SQL und stellt oft eine Alternative zum SAS Datenschritt dar.

Ausschnitt aus der Syntax der SELECT Anweisung mit relevanten Elementen für die Verwendung von `DISTINCT` und zur Erstellung von Makrovariablen:

```
PROC SQL <_METHOD> <_TREE> <NOPRINT>;
  SELECT DISTINCT *|var1 <,...varN>
    INTO :makrovar1 <,...:makrovarN> <SEPARATED BY zeichen>
    <NOTRIM>
    FROM bibliothek.tabelle
    <WHERE bedingung> ... ;

  SELECT ...
QUIT;
```

Anwendung von DISTINCT

`DISTINCT var1 <,...varN)` listet alle unterschiedlichen Werte einer Variablen var1 oder einer Kombination von N Variablen auf. Jeder Wert der Variable wird nur einmal berücksichtigt, d.h. Duplikate werden herausgefiltert.
`DISTINCT *` listet alle unterschiedlichen Einträge in einer Tabelle. „`*`“ steht für alle Variablen/Spalten der Tabelle.
Mit `DISTINCT` werden die Werte bereits aufsteigend sortiert zurückgegeben, da `DISTINCT` durch ein `PROC SORT` mit der Option `NODUP/NODUPKEY` realisiert wird. Dabei werden die Daten sortiert und Duplikate entfernt. Die Verarbeitung von SQL Abfragen kann mit den nicht dokumentierten Anweisungsoptionen `_METHOD` und `_TREE` nachvollzogen werden. Für eine SQL Anweisung zeigt `_METHOD` den dazugehörigen Programmablauf und `_TREE` den detaillierten Programmablauf als Baum im Log Fenster an. Der Output von `_METHOD` und `_TREE` kann sehr hilfreich sein, um den Ablauf von SQL Anweisungen besser zu verstehen und auch den eigenen Code zu optimieren.

Beispieldatensatz

Ein einfacher Datensatz soll die Beispiele veranschaulichen.

```
PROC SQL;
  CREATE TABLE patient /* leere tabelle erstellen */
  (
     id NUM,
     vorname CHAR(20),
     nachname CHAR(20)
  );
QUIT;
PROC SQL;
  INSERT INTO patient
  VALUES (100, 'Harald', 'Schmidt')
  VALUES (205, 'Martin', 'Haller')
  VALUES (378, 'Diana', 'Jung')
  VALUES (425, 'Eva', 'Hein')
  VALUES (545, 'Florian', 'Merk')
  VALUES (689, 'Martin', 'Kerber')
  VALUES (700, 'Barbara', 'Allers')
  VALUES (889, 'Marion', 'Gran');
QUIT;
```

Tabelle 1: Beispieldatensatz

id	vorname	nachname
100	Harald	Schmidt
205	Martin	Haller
378	Diana	Jung
425	Eva	Hein
545	Florian	Merk
689	Martin	Kerber
700	Barbara	Allers
889	Marion	Gran

Beispiele

Beispiel 1 – Einfaches Beispiel mit DISTINCT

```
PROC SQL; /*1*/
  SELECT DISTINCT vorname
    FROM patient;
QUIT;
```

vorname
Barbara
Diana
Eva
Florian
Harald
Marion
Martin

`/*1*/`: Es werden nur unterschiedliche Werte der Variablen `vorname` (aufsteigend) zurückgeliefert. Der Name Martin wird nur einmal aufgelistet.

```
PROC SQL; /*2*/
  SELECT DISTINCT id,vorname
    FROM patient;
QUIT;
```

id	vorname
100	Harald
205	Martin
378	Diana
425	Eva
545	Florian
689	Martin
700	Barbara
889	Marion

`/*2*/`: Es werden nur unterschiedliche Werte der Variablenkombination `id,vorname` , aufsteigend nach `id` und `vorname`, zurückgeliefert. Der Name Martin wird nun zweimal aufgelistet, da er jeweils eine andere `id` besitzt.

Beispiel 2A – Erstellung von Makrovariablen
In diesem Beispiel soll der Output von `DISTINCT` weiterverarbeitet werden, indem ein Array von Makrovariablen, d.h. eine Liste von Makrovariablen, erzeugt wird.

```
PROC SQL NOPRINT; /*1*/
     SELECT DISTINCT nachname INTO :value1-:value&sysmaxlong
     FROM patient;
QUIT;

%LET value_num = &sqlobs; /*2*/

%PUT *&sqlobs* *&value1* *&value2* *&value3*...*&&value&sqlobs*;
```

Output im Log Fenster:
```
*8* *Schmidt* *Haller* *Jung*...*Gran*
```

`/*1*/`: Jeder unterschiedliche Wert der Variable `nachname` wird in einer eigenen Makrovariablen gespeichert, d.h. es entsteht ein Array von Makrovariablen (Abbildung 1). `&sysmaxlong` gibt den maximalen Integerwert wieder (32-bit Windows: 2.147.483.647). Es wird trotzdem nur die benötigte Anzahl an Makrovariablen erstellt.

`/*2*/`: `&sqlobs` gibt die Anzahl der Zeilen, die durch die letzte SQL Anweisung verarbeitet wurden, wieder. Das entspricht bei der SELECT Anweisung der Anzahl der ausgegebenen Zeilen und hier auch der Anzahl der erstellten Makrovariablen.

value1	value2	value3	value4	value5	value6	value7	value8
Schmidt	Haller	Jung	Hein	Merk	Kerber	Allers	Gran

Abbildung 1: Beispiel 2A – Array von Makrovariablen

Hinweise:

- Durch die Verwendung von `&sysmaxlong` ist es nicht notwendig ein `SELECT COUNT()` als erste Abfrage auszuführen, um die benötigte Anzahl an Makrovariablen zu ermitteln. Mit `&sqlobs` erhält man die Anzahl an erstellten Makrovariablen.
- Die Werte der Variable `nachname` werden ohne überschüssige Leerzeichen in die einzelnen Makrovariablen eingetragen. Falls dies nicht erwünscht ist, dan kann `NOTRIM` am Ende der `INTO` Anweisung gesetzt werden.

Beispiel 2B – Erstellung von Makrovariablen
In diesem Beispiel wird ein Makrovariablen-Array erzeugt, d.h. eine einzige Makrovariable, die die gesamte Liste von Werten enthält.

```
PROC SQL NOPRINT; /*1*/
  SELECT DISTINCT nachname INTO :var_values SEPARATED BY ','
    FROM patient;
QUIT;
%PUT *&var_values*;
```

Output im Log Fenster:
```
*Schmidt,Haller,Jung,Hein,Merk,Kerber,Allers,Gran*
```

`/*1*/`: Jeder unterschiedliche Wert der Variable `nachname` wird in eine einzige Makrovariable `varvalues` hintereinander und getrennt durch ein „ , “ eingetragen, d.h. es entsteht ein Makrovariablen-Array (Abbildung 2).

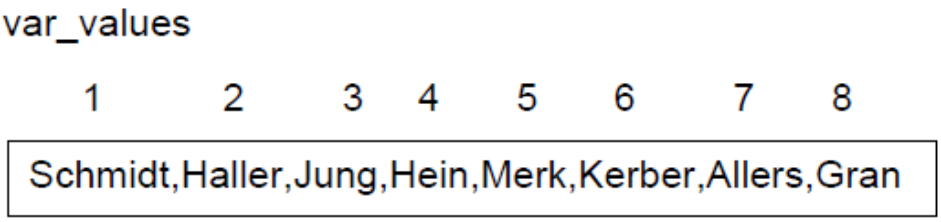

Abbildung 2: Beispiel 2B – Makrovariablen-Array

Ist die Ausgabe mit Anführungszeichen gewünscht, so ist dies mit der Funktion `QUOTE()` möglich:

```
PROC SQL NOPRINT; /*1*/
  SELECT DISTINCT QUOTE(STRIP(nachname)) INTO :var_values SEPARATED
    BY ','
    FROM patient;
QUIT;
%PUT *&var_values*;
```

Output im Log Fenster:

```
*"Allers","Gran","Haller","Hein","Jung","Kerber","Merk","Schmidt"*
```

`/*1*/:` Anführungszeichen

Hinweis:
Überschüssige Leerzeichen müssen vor der Anwendung der `QUOTE()` Funktion mit der Funktion `STRIP()` entfernt werden.

Verarbeitung von DISTINCT

Beispiel 3 – Output von `_METHOD` und `_TREE`

```
PROC SQL _METHOD _TREE;
  SELECT DISTINCT nachname
    FROM patient;
QUIT;
```

Output im Log Fenster für `_METHOD`:

```
NOTE: SQL execution methods chosen are:
      sqxslct *3*
          sqxuniq *2*
              sqxsrc( WORK.PATIENT ) *1*
```

`sqx` im Output von `_METHOD` steht für **SQL** Execution code. Der Output wird von rechts nach links und von unten nach oben gelesen. Die zum Output hinzugefügten Nummern `*1*` bis `*3*` geben die Verarbeitungsreihenfolge an.

`sqxsrc`	kennzeichnet die Quelltabelle
`sqxuniq`	weißt darauf hin, dass `DISTINCT` mit einem `PROC SORT` mit der Option `NODUP/NODUPKEY` umgesetzt wird
`sqxslct`	zeigt die Ausführung einer `SELECT` Anweisung an

Output im Log Fenster für `_TREE`:

```
Tree as planned.
                                  /-SYM-V-(patient.nachname:3 flag=0001)
                        /-OBJ----|    *6*
              /-UNIQ---|   *10*
             |    *11* |                       /-SYM-V-(patient.nachname:3 flag=0001)
             |         |           /-OBJ----|    *2*
             |         |--SRC----|    *5*
             |         |   *9*   \-TABL[WORK].patient opt=''
             |         |--empty-      *4*
             |         |   *8*                 /-SYM-V-(patient.nachname:3 flag=0001)
             |         |           /-ASC----|    *1*
             |          \-ORDR---|   *3*
 --SSEL---|             *7*
    *12*
```

Der Output von `_TREE` wird ebenfalls von rechts nach links und von unten nach oben gelesen. Von rechts nach links werden Anweisungen/Informationen zusammengefasst und eine Ebene weitergegeben. Die zum Output hinzugefügten Nummern `*1*` bis `*12*` geben die Reihenfolge an. Der Output kann folgendermaßen interpretiert werden:

`SYM-V`	selektieren von Variablen aus einer Tabelle. Die Nummer deutet auf die Reihenfolge der Variablen in der Tabelle hin, d.h. ob erste, zweite, dritte ... Variable in der Tabelle. `Flag` ist für interne Weiterverarbeitung vorgesehen
`ASC`	enthält die Information, nach welcher Spalte auf der rechten Seite aufsteigend sortiert werden soll
`TABL[WORK]`	deutet auf die entsprechende Tabelle in der Bibliothek `WORK` hin. Datenschrittoptionen werden in `opt=` vermerkt
`OBJ`	fasst zusammen, welche Variablen eine Ebene höher gereicht werden
`ORDR`	fasst die Information zusammen, nach welchen Spalten wie sortiert werden soll. Diese Information wird eine Ebene weitergegeben an `UNIQ`, welches dann erst das Sortieren durchführt
`empty`	Platzhalter für weitere mögliche Anweisungen für den SQL Optimierer
`SRC`	fasst die Datenquelle zu einem Ergebnisobjekt zusammen
`UNIQ`	eliminieren von Duplikaten, wird durch ein `PROC SORT` mit der Option `NODUP/NODUPKEY` implementiert
`SSEL`	weißt auf eine SQL Abfrage vom Typ `SELECT` hin

Hinweis:

- Der hier beschriebene Output der Optionen `_METHOD` und `_TREE` bezieht sich auf SAS 9.1 und kann von anderen SAS Versionen abweichen.
- `_METHOD` und `_TREE` bieten eine exzellente Möglichkeit die Verarbeitung von SQL Anweisungen nachzuvollziehen. So können sie z.B. genutzt werden, um sich den gewählten Algorithmus zur Umsetzung eines SQL Joins anzeigen zu lassen (z.B. Hash Join, Index Join, Sort Merge Join oder Step Loop Join).

Kurzes Fazit

`PROC SQL` und `DISTINCT` bieten eine flexible und kompakte Vorgehensweise, um Wertelisten als Arrays von Makrovariablen und auch als Makrovariablen-Arrays zu erstellen. Anwendungsszenarien sind zum Beispiel das Auflisten aller unterschiedlichen Werte in einer `WHERE IN (...)` Anweisung, sowie in `WHERE ANY (...)` oder `WHERE ALL (...)` Anweisungen in SQL. Obwohl die Anweisungsoptionen `_METHOD` und `_TREE` nicht dokumentiert sind, bieten sie trotzdem die Möglichkeit etwas hinter die Kulissen von `PROC SQL` zu schauen und die Schritte, die während der Ausführung einer SQL Anweisung ablaufen, genauer zu verstehen. Dieses Verständnis ist hilfreich, um die Performanz der eigenen SQL Anweisungen verbessern zu können.

Literatur

[1] SAS Institute Inc., 2004. SAS® 9.1 SQL Procedure User's Guide. Cary, NC: SAS Institute Inc.

[2] Long, Stuart and Heaton, Ed (2008), Using the SAS DATA Step and PROC SQL to Create Macro Arrays Proceedings of SAS Global Forum 2008 San Antonio, Texas, USA.

[3] Lavery, Russ (2005), The SQL Optimizer Project: _Method and _Tree in SAS®9.1, Proceedings of the Thirtieth Annual SAS® Users Group International Conference Philadelphia, Pennsylvania, USA.

3 Der Data Step Debugger

(Grischa Pfister)

Wenn ein Data Step nicht das tut, was er soll, greifen Generationen von SAS-Anwendern auf PUT-Anweisungen zurück, um den logischen oder datenseitigen Fehler zu finden. Das hat allerdings den Nachteil, dass der Original-Code des Programms verändert werden muss. Dabei gibt es seit langem eine eingebaute Alternative, den Data Step Debugger. Dieser wird durch die Option „/ DEBUG“ im Data Statement zugeschaltet und bietet eine Vielzahl an Möglichkeiten, um den Programmfluss verfolgen und Variablen überwachen zu können.

Wird ein Data Step im Debug-Modus abgeschickt, werden zwei zusätzliche Fenster geöffnet. Das eine zeigt den Code des Data Steps an und markiert die jeweils aktive (noch nicht ausgeführte) Zeile, das andere zeigt das LOG des Debuggers und enthält eine Kommandozeile, in die die Debugger-Anweisungen geschrieben werden. Mit jedem Druck auf die Enter-Taste wird der Data Step zeilenweise abgearbeitet.
Durch verschiedene BREAK-Anweisungen können Haltepunkte gesetzt werden, im einfachsten Falle in einer konkreten Zeile des Data Steps. Für Do-Loops gibt es die Möglichkeit, nach dem n-ten Schleifendurchlauf anzuhalten, der Haltepunkt kann aber auch an eine Bedingung geknüpft werden. Das GO Kommando sorgt dafür, dass der Data Step bis zum nächsten Haltepunkt abgearbeitet wird. Alternativ kann für Variablen eine Überwachung eingeschaltet werden. Wenn sich der Wert einer mit der WATCH-Anweisung markierten Variable ändert, werden alter und neuer Wert im Debugger-LOG ausgegeben und die Verarbeitung des Data Step unterbrochen. Haltepunkte und Überwachungen werden mit DELETE wieder gelöscht.
Die EXAMINE-Anweisung gibt den aktuellen Wert einer Variable in das LOG aus, DESCRIBE zeigt die Variablen-Attribute an. Mit der SET-Anweisung können Variablen-Werte gesetzt werden, allerdings können hier keine SAS-Funktionen verwendet werden – der einzige Nachteil bei der Verwendung des Data Step Debuggers. Mit JUMP kann in eine beliebige Zeile des Programms gesprungen werden, z.B. um eine DO-Schleife nochmals zu durchlaufen, das Quit-Kommando beendet den Debugger.
Es gibt noch weitere Kommandos, die hier nicht vorgestellt werden, sondern der Online-Hilfe bzw. der gedruckten SAS-Dokumentation entnommen werden können.

Literatur

[1] SAS 9.1.3 Language Reference: Dictionary, Fifth Edition, S. 1793 ff.

4 Parallelisierung mit MP Connect

(Grischa Pfister)

Die Verarbeitung immer größerer Datenmengen führt dazu, dass trotz schnellerer Hard- und Software die Ladeprozesse großer Datawarehouses immer wieder an die Grenze der verfügbaren Zeitfenster stoßen. Gefragt sind also Mechanismen, die dafür sorgen, dass in der gleichen Zeit ein Mehr an Datenverarbeitung stattfinden kann. Ein Weg dies zu tun, ist die Methode der Parallelisierung, und SAS unterstützt dies schon seit Version 8 mit MP Connect (MP steht für Multi Processor), einer Erweiterung des klassischen SAS/CONNECT Moduls.

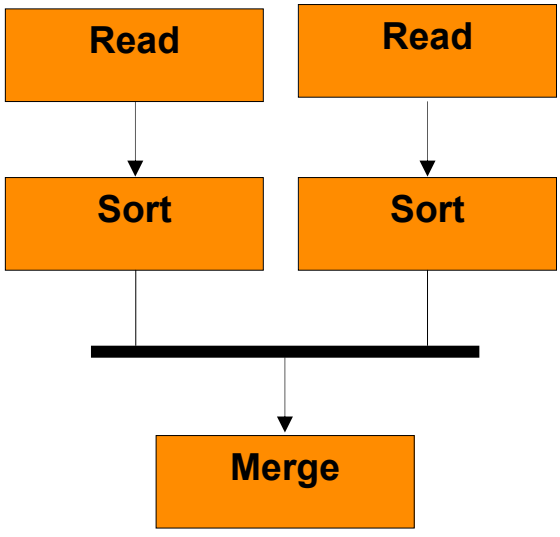

Abbildung 1: Parallele Verarbeitung

Um diese Technik nutzen zu können, muss der eingesetzte Rechner aber bestimmte technische Voraussetzungen erfüllen. Neben mehreren Prozessoren müssen mehrere Platten und meist auch mehrere Kontroller vorhanden sein (Symetric Multiprocessor bzw. SMP Architektur). Anders als bei der sequentiellen Verarbeitung erfolgt die Verarbeitung dann parallel in mehreren getrennten SAS-Sitzungen, die aus einer Haupt-Sitzung aufgerufen und synchronisiert werden.
Die eigentliche Steuerung der Parallelisierung erfolgt durch das Programm, das in der Haupt-Sitzung abgearbeitet wird. Es enthält RSUBMIT-Blöcke für die einzelnen Verarbeitungs-Schritte, die im Gegensatz zum klassischen SAS/CONNECT asynchron abgearbeitet werden. Das bedeutet, die Verarbeitung des Haupt-Programmes geht weiter und wartet nicht darauf, dass der RSUBMIT-Block fertig abgearbeitet ist. Wenn im Hauptprogramm dann ein Schritt folgt, der darauf angewiesen ist, dass asynchron verarbeitete Programmteile fertig sind, kann mit Hilfe der WAITFOR-Anweisung entsprechend synchronisiert werden. Der Programmierer entscheidet also, wie die Parallelisierung eines Programmes konkret ausgestaltet wird.

Ein – abstraktes – Beispiel für ein auf diese Weise parallelisiertes Programm sieht wie folgt aus:

```
Options
  sascmd="D:\Programme\SAS\SAS9\sas.exe -nosyntaxcheck"
  autosignon=yes
;
```

Über ein OPTIONS-Statement wird das Start-Kommando für die asynchron zu startenden SAS-Sitzungen bekanntgegeben, der SIGNON (also der Start) erfolgt automatisch, wenn der RSUBMIT-Block verarbeitet wird.
Anmerkung in eigener Sache: Dieses Start-Kommando war der Grund, warum die Beispiele im Vortrag nicht funktioniert haben. Hier stand nämlich der Pfad zu SAS 9.1.3, geöffnet hatte ich die Beispiele aber unter 9.2 – das konnte nicht funktionieren…

```
*** GP erster RSUBMIT-Block *** ;
Rsubmit process=load01 wait=no persist=no;
  ...
Endrsubmit;
```

Die Option „wait=no“ sorgt dafür, dass die Verarbeitung des RSUBMIT-Blocks asynchron erfolgt, „persist=no“ steuert, dass die SAS-Sitzung anschließend wieder beendet wird. Da hier eine unabhängige SAS-Sitzung gestartet wird, müssen die benötigten Bibliotheken zugeordnet sein, um Tabellen lesen und speichern zu können.

```
*** GP zweiter RSUBMIT-Block *** ;
Rsubmit process=load02 wait=no persist=no;
  ...
Endrsubmit;
```

```
*** GP Synchronisation ***;
Waitfor _all_ load01 load02;

  ...
```

Durch die WAITFOR-Anweisung wird eine Synchronisierung der parallel laufenden RSUBMIT-Blöcke erzwungen, anschließend kann dann die Weiterverarbeitung der getrennt erzeugten Tabellen erfolgen.
Für eine solche Parallelisierung werden insbesondere mehrere Festplatten benötigt, denn jede der asynchronen SAS-Sitzungen liest und schreibt von diesen Platten. Mit nur einer Festplatte würden sich die Prozesse gegenseitig blockieren. Doch auch mit mehreren Festplatten ist der I/O ein limitierender Faktor, weshalb mit SAS 9 eine weitere Komponente in MP Connect eingebaut worden ist, das sog. Piping.

Mit Piping ist gemeint, dass aufeinander folgende Arbeitsschritte wie z.B. Einlesen von Rohdaten mit anschließendem Sortieren der Tabelle ebenfalls nicht sequentiell, sondern verschachtelt passieren können. Während die Daten gelesen werden, können die Beobachtungen, die schon da sind, bereits an die Sortier-Routine übergeben werden – Proc Sort wird also schon aktiv, während der Data Step noch nicht abgeschlossen ist. Dies wird mit Hilfe einer neuen Libname-Engine namens ESOCK verwirklicht, die dafür sorgt, dass die Daten gar nicht erst auf Festplatte gespeichert werden, sondern via Port direkt an einen anderen Prozess übergeben werden, der seinerseits von diesem Port Daten liest. Auch hierfür ein abstrahiertes Beispiel:

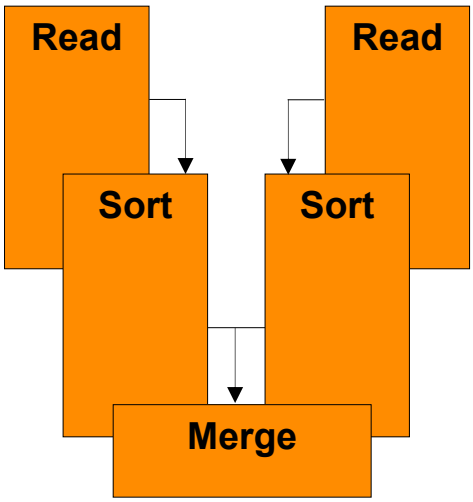

Abbildung 2: Piping

```
Options
  sascmd="D:\programme\sas\sas9\sas.exe -nosyntaxcheck"
  autosignon=yes
;


*** GP Lesen Daten01 *** ;
Rsubmit process=read01 wait=no persist=no;

  Libname out SASESOCK ":5021";

  Data Out.Daten01;
    Infile "daten01.csv";
    ...
  Run;
Endrsubmit;
```

Die Rohdaten für die erste Tabelle werden gelesen und direkt an den Port 5021 geschrieben.

```
*** GP Lesen Daten02 *** ;
Rsubmit process=read02 wait=no persist=no;
  Libname out SASESOCK ":5031";

  Data Out.Daten02;
    ...
  Run;
Endrsubmit;
```

Die Rohdaten für die zweite Tabelle werden ebenfalls gelesen und an einen Port (5031) geschrieben.

```
*** GP Sortieren Daten01 *** ;
Rsubmit process=sort01 wait=no persist=no;
  Libname in SASESOCK ":5021";
  Libname out SASESOCK ":5022";

  Proc Sort data=In.Daten01 out=Out.Daten01;
    By order_date;
  Run;
Endrsubmit;

*** GP Sortieren Daten02 *** ;
Rsubmit process=sort02 wait=no persist=no;
  Libname in SASESOCK ":5031";
  Libname out SASESOCK ":5032";

  Proc Sort data=In.Daten02 out=Out.Daten02;
    By order_date;
  Run;
Endrsubmit;
```

Während die beiden Data Steps asynchron Daten lesen, werden die verarbeiteten Beobachtungen aus dem PDV statt in die Ergebnistabellen direkt an Proc Sort weitergegeben und sortiert. Von hier aus werden wiederum keine Ergebnisdateien erzeugt, sondern es wird an weitere Ports geschrieben (5022 resp. 5032).

```
*** GP Zusammenführen von Daten01 und Daten02 *** ;
Rsubmit process=combine wait=no persist=no;
  Libname out ".";
  Libname in1 SASESOCK ":5022";
  Libname in2 SASESOCK ":5032";

  Data Out.Combine;
    Set In1.Daten01 In2.Daten02;
  Run;
Endrsubmit;
```

Erst im letzten Schritt erfolgt eine Speicherung auf Festplatte, wobei die beiden Tabellen über ein SET-Statement zusammengeführt werden.

Die Piping-Technologie sorgt für eine deutliche Beschleunigung der Verarbeitung, da keine temporären Zwischendateien mehr geschrieben werden. Das kann sich allerdings auch als Achillesferse erweisen, denn wenn ein solches Programm abstürzt, gibt es keinen Zwischenstand mehr, bei dem wieder aufgesetzt werden könnte. Das erreichbare Mehr an Performanz wird also mit einem höheren Risiko bezahlt.

Literatur

[1] SAS/CONNECT 9.1 User's Guide, Part 4 - Compute Services.

5 Automatisches Ermitteln der Achsenbeschriftung

(Heribert Ramroth)

Stellen Sie sich folgende Datensituation vor:
Sie wollen prozentuale Verkaufszahlen / Krankheitshäufigkeiten in mehreren vergleichbaren Situationen abbilden, um die Ergebnisse visuell zu vergleichen.

Ein Beispiel:

- Häufigkeitsverteilungen (prozentual)
  - 4 Quartale
  - 3 Städte

Idee:

- Visuelles Erkennen der Unterschiede in den Quartalen
- Automatisches Ermitteln der Ausprägungen der Städte („*City*", X-Achse)

Der letzte Punkt ist insbesondere dann wichtig, wenn dieses Vorgehen auf viele unterschiedliche Variablen angewendet werden soll, ohne deren *midpoints* jeweils manuell zu ermitteln.

Der Beispieldatensatz:

```
data Bsp;
input Quartal $3. @5 City$2.
@8 value 2.;
datalines;
  I HD 75
  I MA 50
 II MA 25
 II LU 75
III HD 25
III MA 50
III LU 25
 IV LU 50
; run;
```

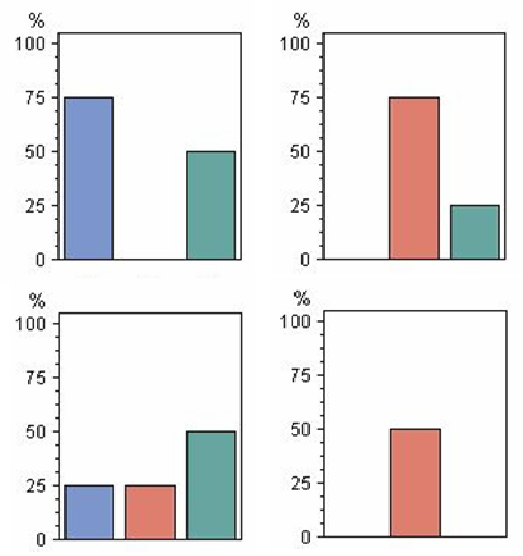

Abbildung 5.1: das Wunschergebnis

Das Problem: Der Standard SAS/GRAPH Code stellt nur 2 unzureichende Möglichkeiten zur Verfügung, die im Folgenden dargestellt werden:

1) by:

```
proc gchart data=bsp;
     vbar city/ patternid= midpoint;
     by Quartal ;
run; quit;
```

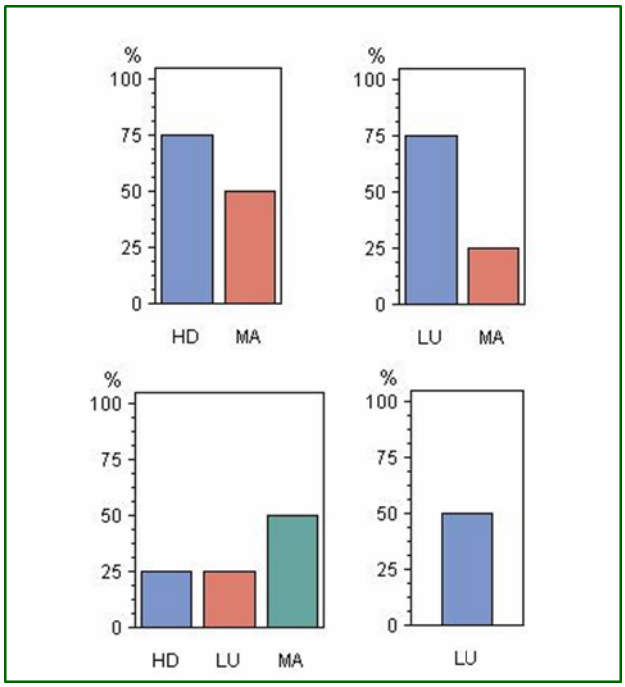

Abbildung 5.2: by statement

2) Option group:

```
proc gchart data=bsp;
     vbar city/ patternid= midpoint group= Quartal;
     by Quartal ;
run; quit;
```

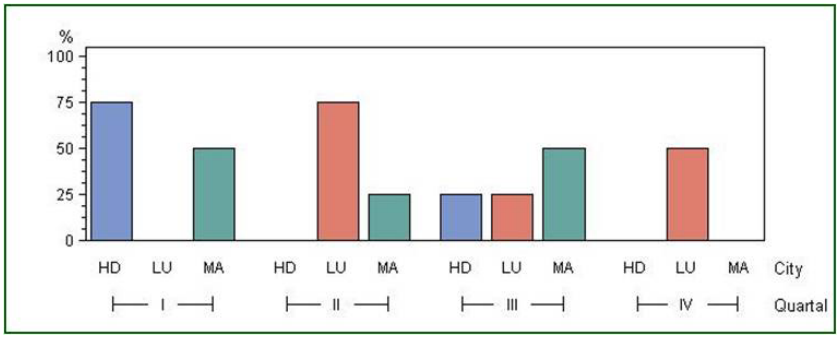

Abbildung 5.3: option group

Die Lösung nach manueller Ermittlung der Ausprägungen der Variablen City lautet wie folgt:

```
proc gchart data=bsp;
   vbar city /   patternid= midpoint   midpoints= "HD" "LU" "MA";
run;
quit;
```

Frage: Wie kommen alle Ausprägungen/Kategorien der Variablen City ins midpoints statement, ohne sie manuell erst zu ermitteln und anschließend ins midpoints statement reinzuschreiben?

Die Makro-Lösung: Man ermittelt alle Ausprägungen „automatisch“.

Der zugehörige Makro-Code lautet wie folgt:

```
%Macro GetGroups(table= ,var= ,mvar= );
/*   table     Name der Quelle (Lib.Tab)
     var       Name der Variable, deren Ausprägungen ermittelt werden
     mvar      Name der Makrovariable, in die das Ergebnis gespei-
               chert wird   */
  %Global &mvar;
  /* Abfrage der distinkten Ausprägungen */
  Proc Sql Noprint;
     Select Distinct &var Into :&mvar Separated By ' " " '
          From &table;
  Quit;
```

```
  /* führendes und schließendes Anführungszeichen */
  %Let &mvar = "&&&mvar";
%Mend;

%GetGroups(table=bsp, var=City, mvar=gr);

axis1 order=(0 to 100 by 25) label=("%");
axis2 label="";

proc gchart data=bsp;
     vbar city/ freq= value   raxis= axis1 maxis= axis2
          patternid= midpoint midpoints = &gr. ;
run; quit;
```

Literatur

[1] SAS Institute Inc., 2004. SAS® 9.1 SQL Procedure User's Guide. Cary, NC: SAS Institute Inc.

6 Eigene Funktionen – jetzt auch im Datenschritt – mit der Prozedur FCMP

(Carina Ortseifen)

Die SAS Software stellt dem Anwender sehr viele vorbereitete Funktionen zur Verfügung, die im Datenschritt und bei manchen Prozeduren, SQL, NLIN, MODEL etc. eingesetzt werden können.
Um eigene Funktionen verwenden zu können, mussten im Datenschritt Makros zum Einsatz kommen oder mittels Link- und Return-Anweisung verbundene Datenschritt-Abschnitte. Mit SAS Version 9.1.3 steht die Prozedur FCMP zur Verfügung und erlaubt eigene Funktionen zu definieren, die mit den oben erwähnten Prozeduren verwendet werden können.
Mit SAS 9.2 ist nun die Lücke zum Datenschritt geschlossen und die mittels der Prozedur FCMP definierten Funktionen können auch im Datenschritt eingesetzt werden.

Mit den folgenden Beispielen möchten wir Ihr Interesse wecken, sich selbst mit diesen neuen Möglichkeiten zu beschäftigen.

Ein erstes Beispiel: Berechnung des Alters

Zunächst wird die neue Funktion Alter mit der Prozedur FCMP definiert:

```
PROC FCMP OUTLIB=sasuser.funktionen.paket;
   FUNCTION alter (geburtsdatum, stichtag);
      RETURN (floor((intck('month',geburtsdatum,stichtag)
            - (day(stichtag) < day(geburtsdatum))) / 12));
   ENDSUB;
RUN;
```

Die Funktion alter wird in der Bibliothek sasuser innerhalb der SAS-Tabelle funktionen im Paket paket abgelegt (Option OUTLIB=).
PROC FCMP OUTLIB=bib.tab.paket;

Die Funktion alter wird mit zwei Argumenten definiert, geburtsdatum und stichtag.
FUNCTION(argument-1, argument-2);

Der Rechenausdruck für das Alter wurde der SAS-Web-Seite „Sample 24808: Accurately Calculating Age with Only One Line of Code“
[http://support.sas.com/kb/24/808.html, 13.03.2009].
Die RETURN-Anweisung liefert das Ergebnis, den Rückgabewert der Funktion. (Es gibt auch Funktionen ohne Rückgabewert. Diese werden mit der Anweisung SUBROUTINE definiert.)
RETURN (ergebnis);

Die Anweisung ENDSUB beendet die Definiton und RUN den Prozedurschritt.

Im zweiten Schritt benötigen Sie nun den Suchpfad, der SAS anzeigt, wo nach Benutzer-eigenen Funktionen gesucht wird (Option CMPLIB=).

```
OPTIONS CMPLIB=sasuser.funktionen;
```

Und schließlich folgt der Datenschritt, in dem die neue Funktion eingesetzt wird.

```
DATA daten;
   INPUT geburt ddmmyy10.;
   heute="05MAR2009"d;
   altr=alter(geburtsdatum,stichtag);
   FORMAT geburt heute ddmmyy10.;
   DATALINES;
05.03.2009
05.03.2008
06.03.2008
01.01.2008
01.01.2000
01.01.1960
;
```

```
TITLE "Alter in Jahren am 05.03.2009";
PROC PRINT DATA=daten;
   VAR geburt altr;
RUN;
```

Was (unter Windows XP und SAS 9.2) zu folgendem Ergebnis führt:

```
Alter in Jahren am 05.03.2009

 Obs         geburt     altr

  1      05/03/2009        0
  2      05/03/2008        1
  3      06/03/2008        0
  4      01/01/2008        1
  5      01/01/2000        9
  6      01/01/1960       49
```

Anmerkung:
Glücklicherweise können Sie keine SAS-eigenen Funktionen überschreiben, wie das folgende Beispiel zeigt:

```
PROC FCMP OUTLIB=sasuser.funktionen.paket;
   FUNCTION normal(zahl);
     RETURN (zahl*2);
   ENDSUB;
RUN;
OPTIONS CMPLIB=sasuser.funktionen;
data zufall;
   do i=1 to 10;
      x=normal(i);
       output;
   end;
run;
proc print data=zufall;
run;
```

Die Funktion NORMAL() erzeugt Standardnormalverteilte Pseudozufallszahlen. Obiger Prozedur FCMP-Schritt definiert eine Funktion Normal, die das Argument verdoppeln soll. Im Ergebnis und der Tabelle Zufall sehen wir aber, dass SAS Zufallszahlen erzeugt und nicht verdoppelt. Was uns als Anwender davor bewahrt, SAS-eigene Funktionen absichtlich oder unabsichtlich zu überschreiben.

Leider kommt es dabei zu keinerlei Warnung oder Hinweis, dass die eigene Funktion nicht zum Einsatz kommt.

Eine, wenn auch unvollständige Liste der Funktionen in SAS 9.1.3 findet man unter http://support.sas.com/techsup/technote/ts486.pdf

Subroutinen – Funktionen ohne Rückgabewert

Analog zu den Funktionen können Sie sich mit Hilfe der Prozedur FCMP auch eigene Unterroutinen schreiben, die ähnlich wie die Call Routinen keine Rückgabewerte haben.

```
PROC FCMP OUTLIB=sasuser.funktionen.paket;
    SUBROUTINE inverse (in, inv);
     OUTARGS inv;
      IF in=0 THEN inv=.;
      ELSE inv=1/in;
   ENDSUB;
RUN;
```

Anstelle der Anweisung FUNCTION steht hier SUBROUTINE. Die Anweisung RETURN ist optional, d.h. sie kann wie in obigem Beispiel auch weggelassen werden.

Die Beispielroutine inverse soll das Inverse einer Zahl berechnen und – falls 0 eingegeben wird, einen fehlenden Wert. zurückgeben.
Das Ergebnis dieser Routine wird nicht mit Return ausgegeben, sondern in eine Variable gesteckt, die als Argument mit übergeben wurde, inv. Damit diese Variable Veränderungen speichert, muss die Anweisung OUTARGS gesetzt werden.

```
DATA _NULL_;
   INPUT a @@;
   b=.;
   CALL inverse(a, b);
   PUT a b;
   DATALINES;
1 . 2 0
;
```

Im Log-Fenster erhalten wir:

```
1 1
. .
2 0.5
0 .
```

Geltungsbereiche der Variablen

Im Gegensatz zu Makrovariablen sind die innerhalb der Funktionen und Subroutinen definierten Variablen stets lokale Variablen, d.h. die Geltungsbereiche sind absolut gegeneinander abgegrenzt und Überschreibungen sind dadurch ausgeschlossen, wie folgendes einfache Beispiel zeigen soll:

```
PROC FCMP OUTLIB=sasuser.funktionen.varscope;

   SUBROUTINE subB();
      x="subB";
      PUT "In subB:" x=;
   ENDSUB;

   SUBROUTINE subA();
      x=10;
       call subB();
       put 'In subA:' x=;
   ENDSUB;
RUN;

OPTIONS CMPLIB=sasuser.funktionen;
DATA _NULL_;
   x=99;
   CALL subA();
   PUT "Im Datenschritt:" x=;
RUN;
```

Innerhalb des aufrufenden Datenschritts wird die Variable x definiert und auf 99 gesetzt. Die beiden Subroutinen subA und subB verändern diese Variable im Wert und auch im Variablentyp, ohne dass dabei Probleme auftreten:

```
In subB: x=subB
In subA: x=10
Im Datenschritt:x=99
```

Was noch geht

Die weiteren Möglichkeiten der Funktionen sollen hier nur noch aufgezählt werden. Für Details verweisen wir auf die Beispielprogramme auf der Begleit-CD und die Literatur.

- Rekursion
  Sie können Funktionen rekursiv aufrufen, um z.B. alle k aus n Permutationen zu erzeugen (Beispielprogramm coFCMP4.sas).

- Dynamische Arrays
  Wenn man nicht genau vorhersagen kann, wie viele Arrayfelder notwendig sein werden, können dynamische Arrays eingesetzt werden. Z.B. um Verzeichnisin-

halte auszulesen (Beispielprogramm coFCMP4.sas).

- Weitere Einsatzbereiche der eigenen Funktionen
  Mit FCMP definierte Funktionen können auch in Where-Anweisungen in Prozedurschritten und mit der Prozedur SQL eingesetzt werden (Beispielprogramm coFCMP6.sas).

- FCMP-Routinen sind auch in Proc Report-Compute-Blöcken einsetzbar.

- Die Inlib-Option erlaubt Zugriff auf vorhandene Pakete und auch C-Programme (Proc Proto).

- Mit der Funktion SOLVE können Routinen implizit innerhalb des gleichen FCMP-Schritts aufgelöst werden.

Was anders geht

Einige Anweisungen funktionieren beim Definieren von eigenen Funktionen mit der FCMP-Prozedur anders als im normalen Datenschritt. Dazu gehören die

- PUT-Anweisung: Sie kann nicht zum Formatieren eingesetzt werden, nur zum Debuggen, dabei sind aber Funktionen erlaubt, wie z.B. PUT (x/100) (sqrt(y)/2);

- IF-Anweisung: x = IF y < 100 THEN 1 ELSE 0;

- Arrays können mit eckigen und auch geschweiften Klammern definiert warden.

Was nicht oder noch nicht geht

Folgende Anweisungen können nicht für die Definition von eigenen Funktionen verwendet werden: Data, Set, Merge, Update, Modify, Input, Output, Infile, Do-Anweisung mit Textwerten, Data Step Debugger.

Und noch nicht nutzbar sind die eigenen Funktionen in Kombination mit %sysfunc:

```
data _null_;
   put "Das Alter beträgt
      %sysfunc(alter('01JAN1960'd,'30JUN2008'd)) Jahre.";
run;
```

Eine direkte Berechnung der Funktion alter (Definition siehe oben) bei der Ausgabe mit PUT geht noch nicht, sondern nur in zwei Schritten zerlegt:

```
data _null_;
   alter=alter("01JAn1960"d,"30JUN2008"d);
   put "Das Alter beträgt " alter "Jahre.";
```

```
run;
```

Für spätere Versionen ist dieses Feature jedoch angekündigt.

Literatur

[1] SAS Online Doc
http://support.sas.com/documentation/cdl/en/proc/59565/PDF/default/proc.pdf

[2] Jason Secosky: User Written Data Step Functions
http://support.sas.com/rnd/base/datastep/user-written-functions.pdf
http://www2.sas.com/proceedings/forum2007/008-2007.pdf

7 Endlich richtig sortieren – Option Sortseq der Prozedur SORT

(Carina Ortseifen)

Das Sortieren von Tabellen sollte eigentlich ein leichtes sein und zu den grundlegenden Tugenden einer analytischen Software gehören. Doch im Zuge einer immer globaler werdenden Welt, in der man nicht nur mit der englischen, sondern auch mit Deutsch, Schwedisch, Japanisch und Chinesisch zu tun hat, um nur ein paar der vorkommenden Sprachen zu nennen, wird schnell klar, dass Sortieren vielleicht doch nicht so einfach ist.

Wir wollen folgende Begriffe alphabetisch sortieren:
Esel eins essen dass daß älter USA und übung Übung alt 0123 123 aelt ub Alter

Dazu lesen wir diese in eine SAS-Tabelle szene ein und sortieren nach der Variable stichwort:

```
Proc Sort Data=szene Out=test;
  By Stichwort;
Run;
```

Und erhalten die folgende Liste:

```
Obs    stichwort

  1      0123
  2      123
  3      Alter
  4      Esel
  5      USA
  6      aelt
```

```
   7      alt
   8      dass
   9      daß
  10      eins
  11      essen
  12      ub
  13      und
  14      Übung
  15      älter
  16      übung
```

Nach den Ziffern 0, 1 usw. erscheinen die mit Großbuchstaben beginnenden Worte von A bis U, dann die mit Kleinbuchstaben beginnenden und schließlich die mit einem Umlaut beginnenden Buchstaben. Sehr schön und eindeutig, aber für uns Deutsche eher unbrauchbar.

Mit SAS 9.2 gibt es die neue Option SORTSEQ=LINGUISTIC, welche ein linguistisches Sortieren erlaubt. Und die Suboption COLLATION=PHONEBOOK passt genau auf die Bedürfnisse der Deutschen, die Worte analog zum Telefonbuch zu sortieren. D.h. Klein- und Großbuchstaben-Wörter sind nicht strikt getrennt und ä wird aufgelöst in ae und zwischen ad und af einsortiert.

Das erweiterte Programm hat dann folgende Form:

```
proc sort data=szene out=test
     Sortseq=Linguistic(collation=phonebook);;
   by stichwort;
run;
```

und liefert folgendes Ergebnis:

```
Obs    stichwort

   1      0123
   2      123
   3      aelt
   4      älter
   5      alt
   6      Alter
   7      dass
   8      daß
   9      eins
  10      Esel
  11      essen
```

```
  12      ub
  13      übung
  14      Übung
  15      und
  16      USA
```

Eine weitere schöne Suboption ist NUMERIC_COLLATION=ON, womit Hausnummern von Straßennamen und andere Zahlen innerhalb von Textwerten korrekt sortiert werden können:

```
data liste;
   input name $ 1-20;
   datalines;
X-Weg 2
X-Weg 120
X-Weg 10
;
proc sort data=liste out=test
      sortseq=linguistic(collation=phonebook
      numeric_collation=on);
   by name;
run;
```

Mit dem Ergebnis:

```
Obs      name

  1     X-Weg 2
  2     X-Weg 10
  3     X-Weg 120
```

Literatur

[1] Online-Hilfe zu Proc Sort
http://support.sas.com/documentation/cdl/en/proc/59565/HTML/default/a000057941.htm

[2] Linguistic Collation: Everyone Can Get What They Expect. Sensible Sorting for Global Business Success. SAS Technical Paper.
http://support.sas.com/resources/papers/linguistic_collation.pdf

8 Checkpoint und Restart Modus

(Grischa Pfister)

Für die Belieferung von Datawarehouses oder andere lang laufende Arbeitsschritte wird SAS häufig im Batch verwendet. Zumeist nachts in bestimmten Zeitfenstern laufen diese SAS-Prozesse, am nächsten Tag prüft dann ein zuständiger Mitarbeiter, ob alles geklappt hat. Leider kann es hier immer wieder zu Problemen kommen, weil z.B. Input-Tabellen noch nicht vollständig waren oder der temporäre Arbeitsspeicher nicht ausgereicht hat. In diesen Fällen wird die Batch-Verarbeitung abgebrochen und muss dann erneut gestartet werden.
Bisher war es so, dass das ganze Programm ein zweites Mal abgearbeitet werden musste. Neben einer Reihe weiterer Implikationen bedeutet das, dass der Zeitaufwand für den Programmdurchlauf noch einmal mindestens genauso groß ist. Mit Version 9.2 gibt es jetzt eine Neuerung, die für solche Fälle eine Lösung anbietet – der Checkpoint und Restart Modus.
Der Checkpoint Modus wird über Optionen aktiviert und sorgt dann dafür, dass in einem Katalog mitprotokolliert wird, welche Arbeitsschritte korrekt durchlaufen wurden, bevor es zu dem Fehler kam. Nach Abbruch des Programms kann SAS dann erneut gestartet werden, diesmal im Restart Modus. Aus dem Katalog wird ausgelesen, welche Programmteile erfolgreich gelaufen waren, und ab welcher Stelle wieder neu aufgesetzt werden muss. Enthaltene Makros werden erneut kompiliert, Makrovariablen allerdings nicht wieder mit den Laufzeitwerten befüllt. Über Optionen kann auch gesteuert werden, dass die Work-Bibliothek auf das Verzeichnis der abgebrochenen SAS-Session gesetzt wird – so sind auch die erfolgreich erstellten temporären Tabellen aus der vorherigen Sitzung noch nutzbar. Das globale Statement „Checkpoint Execute_Always“ steuert, dass der nachfolgende Programmabschnitt auf jeden Fall erneut ausgeführt wird. Eine Übersicht über die Funktionsweise des Checkpoint und Restart Modus sowie über die hierfür neu eingeführten Optionen finden sich in der Language Reference im entsprechenden Unterkapitel zur Fehlerbehandlung – generell eine interessante Lektüre.

Literatur

[1] SAS 9.2 Language Reference: Concepts, S. 108 ff.