SAS DATA Step - Optionen vs. Anweisungen

Biljana Gigić
Katharina Buck Cornelia M. Ulrich
Abteilung Präventive Onkologie
Nationales Centrum für Tumorerkrankungen (NCT)
Deutsches Krebsforschungszentrum (DKFZ) Heidelberg
Im Neuenheimer Feld 460
69120 Heidelberg
biljana.gigic@nct-heidelberg.de
katharina.buck@nct-heidelberg.de neli.ulrich@nct-heidelberg.de

Zusammenfassung

In diesem Beitrag wird die Verarbeitung eines SAS DATA Steps während einer KEEP-, DROP-, IF-, WHERE- und RETAIN-Anweisung bzw. Option beleuchtet. Verschiedene Anwendungsmöglichkeiten von KEEP und DROP sowie die bedeutenden Unterschiede der IF- und WHERE-Anweisung werden dargestellt. Dabei sollen anhand von Beispielen die Datenverarbeitung innerhalb der Kompilierungs- und Ausführungsphase eines DATA Steps demonstriert und die daraus resultierenden Outputs gegenüber gestellt werden. Das Verständnis dieser Prozesse erleichtert den richtigen Umgang bei der Erzeugung und Auswahl von Variablen und Datensätzen innerhalb eines DATA Steps.
Erläuterung: Ein SAS DATA Step wird in zwei aufeinanderfolgenden Phasen verarbeitet (Kompilierungs- und Ausführungsphase). Während der Kompilierungsphase werden der Program Data Vector (PDV) und der beschreibende Teil der neu zu erzeugenden Datei generiert. In der Ausführungsphase wird die Output-Datei erzeugt. Abhängig von den verwendeten Anweisungen und Optionen werden die einzelnen Beobachtungen im PDV gespeichert und nach jedem Durchgang des DATA Steps in die Output-Datei transferiert.

Schlüsselwörter: Data Step, Program Data Vector, Kompilierungsphase, Ausführungsphase, KEEP, DROP, WHERE, Subsetting-IF, RETAIN

1 Verarbeitungsphasen eines SAS DATA Steps

Ein SAS DATA Step wird grundsätzlich in zwei aufeinanderfolgenden Phasen verarbeitet: Kompilierungs- und Ausführungsphase [1].

1. Mit dem Submitten (-Klick) des DATA Steps beginnt die **Kompilierungsphase**. Dabei wird die **Syntax** des DATA Steps geprüft, ein **Program Data Vector** (PDV) und der **beschreibende Teil** der neu zu erzeugenden SAS-Datei generiert. Mit dem RUN-Statement endet die Kompilierung.
2. Es folgt die **Ausführungsphase**. Abhängig von den verwendeten Anweisungen und Optionen wird ein **DATA Step pro Beobachtung** ausgeführt. Dabei werden die einzelnen Beobachtungen im PDV gespeichert und nach jedem Durchgang des DATA Steps in die neue SAS-Datei transferiert.

Anhand eines Beispiels sollen die Verarbeitungsphasen verdeutlicht werden. Gegeben sei die SAS-Datei **work.Projekte**, die die Variablen Name (Name der Mitarbeiter), **P1_t**, **P2_t**, **P3_t** (Projektzeiten 1 bis 3) und **P1_S**, **P2_S**, **P3_S** (Projektstatus 1 bis 3) beinhaltet.

work.Projekte

	Name	P1_t	P2_t	P3_t	P1_S	P2_S	P3_S
1	Back	21	.	25	done		done
2	Bayer	16	.	26	done		done
3	Becker	.	25	22		done	done
4	Dieter	.	27	22		tbd	tbd
5	Faust	.	25	23		done	tbd
6	Franke	14	.	22	tbd		tbd
7	Mayer	.	25	23		tbd	tbd
8	Müller	17	.	27	tbd		tbd
9	Sauer	12	.	27	tbd		tbd
10	Schmidt	15	.	22	tbd		tbd

Die Datei **work.Mitarbeiterzeiten** soll erzeugt und dabei die Summe **sumP_t** über die Projektzeiten **P1_t** bis **P3_t** gebildet werden.

```
data Mitarbeiterzeiten;❶
  set Projekte;❷
  sumP_t=sum(P1_t,P2_t,P3_t);❸
run;❹
```

1.1 Kompilierungsphase

Mit dem DATA-Statement ❶ beginnt die Kompilierungsphase. Im nächsten Schritt ❷ werden aus der SAS-Datei **work.Projekte** alle Variablen nacheinander im PDV angelegt. In Schritt ❸ wird die Variable **sumP_t** im PDV angelegt. Der PDV führt zwei automatische, temporäre Variablen _N_ und _ERROR_ mit sich. _N_ gibt dabei die Anzahl der Ausführungen des DATA Steps an, _ERROR_ kontrolliert fehlerhafte Daten. Diese werden nicht an die neu zu erzeugende SAS-Datei übergeben, sie dienen lediglich zur Kontrolle der Verarbeitung der Daten [2].

PDV

Name $ 8	P1_t N 8	P2_t N 8	P3_t N 8	P1_S $ 8	P2_S $ 8	P3_S $ 8	sumP_t N 8	_N_ N 8	_ERROR_ N 8
								1	0

Am Ende der Kompilierungsphase ❹ wird der beschreibende Teil der neuen SAS-Datei generiert. Die SAS-Datei selbst beinhaltet zu diesem Zeitpunkt keine Beobachtungen, da der Datenschritt noch nicht ausgeführt wurde.

1.2 Ausführungsphase

Nach der Kompilierungsphase folgt die Ausführungsphase des DATA Steps.

```
data Mitarbeiterzeiten;❶
  set Projekte;❷
  sumP_t=sum(P1_t,P2_t,P3_t);❸
run;❹
```

Mit dem DATA-Statement ❶ wird der PDV initialisiert, d.h. alle Variablen werden zunächst auf Missing gesetzt:

PDV

Name $ 8	P1_t N 8	P2_t N 8	P3_t N 8	P1_S $ 8	P2_S $ 8	P3_S $ 8	sumP_t N 8	_N_ N 8	_ERROR_ N 8
	.	.	.				.	1	0

In Schritt ❷ werden die Werte der Variablen der ersten Beobachtung aus der Datei **work.Projekte** in den PDV geschrieben:

PDV

Name $ 8	P1_t N 8	P2_t N 8	P3_t N 8	P1_S $ 8	P2_S $ 8	P3_S $ 8	sumP_t N 8	_N_ N 8	_ERROR_ N 8
Back	21	.	25	done		done	.	1	0

Im nächsten Schritt ❸ erfolgt die Berechnung der Summe der drei Projektzeiten. Der Wert der Variable **sumP_t** wird in den PDV geschrieben:

PDV

Name $ 8	P1_t N 8	P2_t N 8	P3_t N 8	P1_S $ 8	P2_S $ 8	P3_S $ 8	sumP_t N 8	_N_ N 8	_ERROR_ N 8
Back	21	.	25	done		done	46	1	0

Am Ende des DATA Steps, vor dem RUN-Statement, erfolgen automatisch ein impliziter OUTPUT und ein impliziter RETURN. Mit dem impliziten OUTPUT werden die Werte der Variablen der ersten Beobachtung aus dem PDV in die neue Datei **work.Mitarbeiterzeiten** transferiert.

work.Mitarbeiterzeiten

Name	P1_t	P2_t	P3_t	P1_S	P2_S	P3_S	sumP_t
Back	21	.	25	done		done	46

Durch den impliziten RETURN springt das Programm an den Anfang des DATA Steps. Der zweite DATA Step beginnt mit dem DATA-Statement und der Reinitialisierung des PDV, Schritt ❶. Während die neu zu erzeugende Variable **sumP_t** auf Missing gesetzt wird, behalten die restlichen Variablen, die aus einer SAS-Datei eingelesen werden, den

vorherigen Wert. Zudem wird die Iteration hochgezählt, _N_ wird auf 2 erhöht (d.h. das Programm befindet sich bei der zweiten Beobachtung).

PDV

Name $ 8	P1_t N 8	P2_t N 8	P3_t N 8	P1_S $ 8	P2_S $ 8	P3_S $ 8	sumP_t N 8	_N_ N 8	_ERROR_ N 8
Back	21	.	25	done		done	.	2	0

Mit dem SET-Statement ❷ werden die Werte der zweiten Beobachtung aus der Datei **`work.Projekte`** in den PDV überführt.

PDV

Name $ 8	P1_t N 8	P2_t N 8	P3_t N 8	P1_S $ 8	P2_S $ 8	P3_S $ 8	sumP_t N 8	_N_ N 8	_ERROR_ N 8
Bayer	16	.	26	done		done	.	2	0

In Schritt ❸ wird der Wert der Variable **`sumP_t`** in den PDV geschrieben.

PDV

Name $ 8	P1_t N 8	P2_t N 8	P3_t N 8	P1_S $ 8	P2_S $ 8	P3_S $ 8	sumP_t N 8	_N_ N 8	_ERROR_ N 8
Bayer	16	.	26	done		done	42	2	0

Mit dem impliziten OUTPUT wird die zweite Beobachtung in die SAS-Datei **`work.Mitarbeiterzeiten`** transferiert und das Programm springt erneut an den Anfang des DATA Steps.

`work.Mitarbeiterzeiten`

Name	P1_t	P2_t	P3_t	P1_S	P2_S	P3_S	sumP_t
Back	21	.	25	done		done	46
Bayer	16	.	26	done		done	42

Analog werden alle weiteren Beobachtungen verarbeitet. Wird das End of File (keine weiteren Beobachtungen vorhanden) erreicht, wird das RUN-Statement ❹ ausgeführt. Der DATA Step wird geschlossen und der nächste DATA- oder PROC-Schritt bearbeitet.

2 SAS DATA Step

Das Prinzip der Verarbeitung eines SAS DATA Steps soll in der folgenden Abbildung dargestellt werden [3].

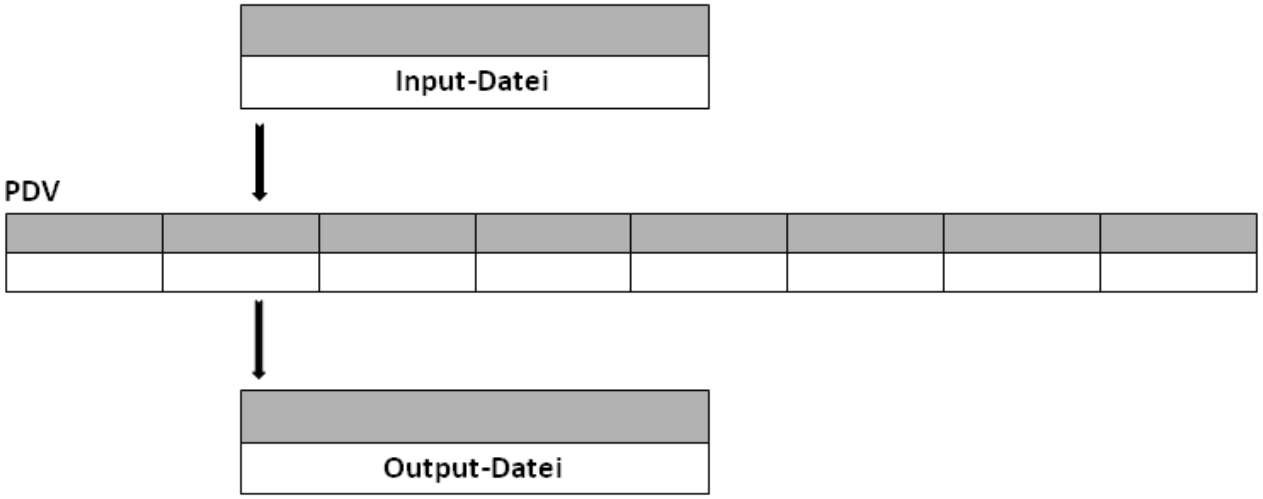

Abbildung 1: SAS DATA Step, Grundprinzip der Verarbeitung

Dabei wird eine Input-Datei (im Beispiel **`work.Projekte`**) über den Program Data Vector (PDV) pro Beobachtung in die Output-Datei (**`work.Mitarbeiterzeiten`**) überführt. Dazwischen können verschiedene Optionen und Anweisungen definiert werden:

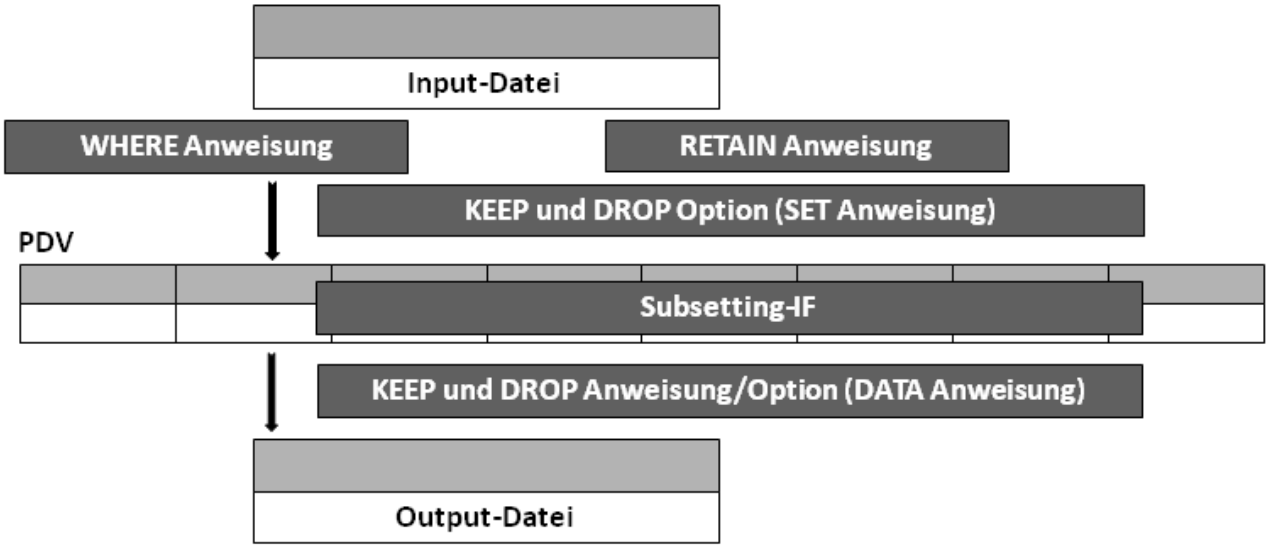

Abbildung 2: SAS DATA Step, Optionen und Anweisungen

Abhängig von der Wahl der Option bzw. Anweisung, wird eine Beobachtung zunächst in den PDV eingelesen und nicht in die Output-Datei transferiert, oder aber erst gar nicht aus der Input-Datei in den PDV eingelesen. Das Gleiche gilt auch für Variablen: Auch hier können zunächst alle Variablen im PDV angelegt werden und erst im nächsten Schritt, nach der Verarbeitung, selektiert in die Output-Datei ausgegeben werden.

Möchte man aber die Verarbeitung von nicht benötigten Variablen vermeiden, können diese anhand von geeigneten Optionen vor dem Einlesen bzw. Anlegen in den PDV selektiert werden.
Diese Unterschiede werden in Kapitel 3 und 4 beschrieben.

3 KEEP und DROP: Optionen vs. Anweisungen

KEEP- und DROP-Optionen und -Anweisungen dienen zur Selektion von Variablen. Die KEEP- und DROP-Option ist sowohl innerhalb der DATA-Anweisung als auch innerhalb der SET-Anweisung gültig. Um effizientere DATA Steps implementieren zu können, ist das Verständnis der Unterschiede essentiell.

```
data OutDataSet1<(keep=Var1 Var2 ...)>
     OutDataSet2<(keep=Var1 Var3 Var4 ...)> ...; /*KEEP-Option*/
  set InDataSet1<(keep=Var1-Var8)>; /*KEEP-Option*/
  <Statement;>
  <keep Var1 Var2 ...;> /*KEEP-Anweisung*/
run;
```

Wird die **KEEP-Option innerhalb der DATA-Anweisung** gesetzt, so werden zunächst alle Variablen aus der Input-Datei im PDV angelegt und bearbeitet, jedoch nur diejenigen an die Output-Datei übergeben, die innerhalb der KEEP-Option definiert wurden (siehe Abbildung).

```
data Mitarbeiterzeiten(keep=Name sumP_t);
  set Projekte;
  sumP_t=sum(P1_t,P2_t,P3_t);
run;
```

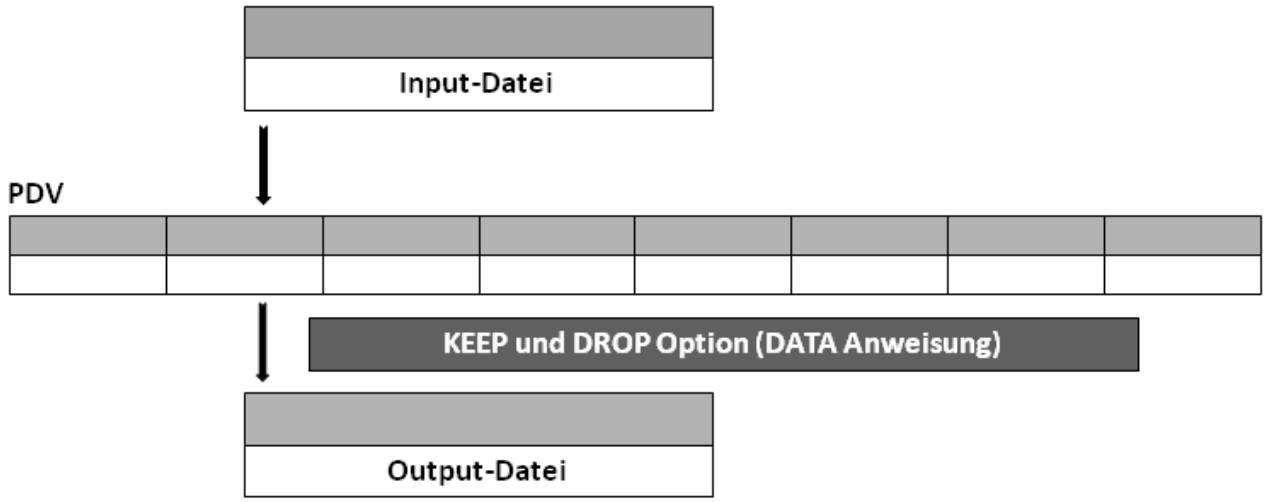

Abbildung 3: KEEP- und DROP-Option innerhalb der DATA-Anweisung

Setzt man die **KEEP-Option innerhalb der SET-Anweisung**, werden nur die Variablen, die innerhalb der KEEP-Option definiert wurden, in den PDV angelegt, verarbeitet und in die Output-Datei ausgeben. In diesem Fall können nicht benötigte Variablen von der Verarbeitung ausgeschlossen werden. Bei sehr großen SAS-Dateien kann das zu einer Verbesserung der Performance führen. Werden Variablen zur Berechnung oder Weiterverarbeitung benötigt, sollen diese allerdings nicht in der neuen SAS-Datei angelegt werden, so müssen sie zunächst in den PDV eingelesen werden. Nachträglich können sie mit einer DROP-Option innerhalb der DATA-Anweisung gelöscht werden.

```
data Mitarbeiterzeiten;
  set Projekte(keep=Name sumP_t);
  sumP_t=sum(P1_t,P2_t,P3_t);
run;
```

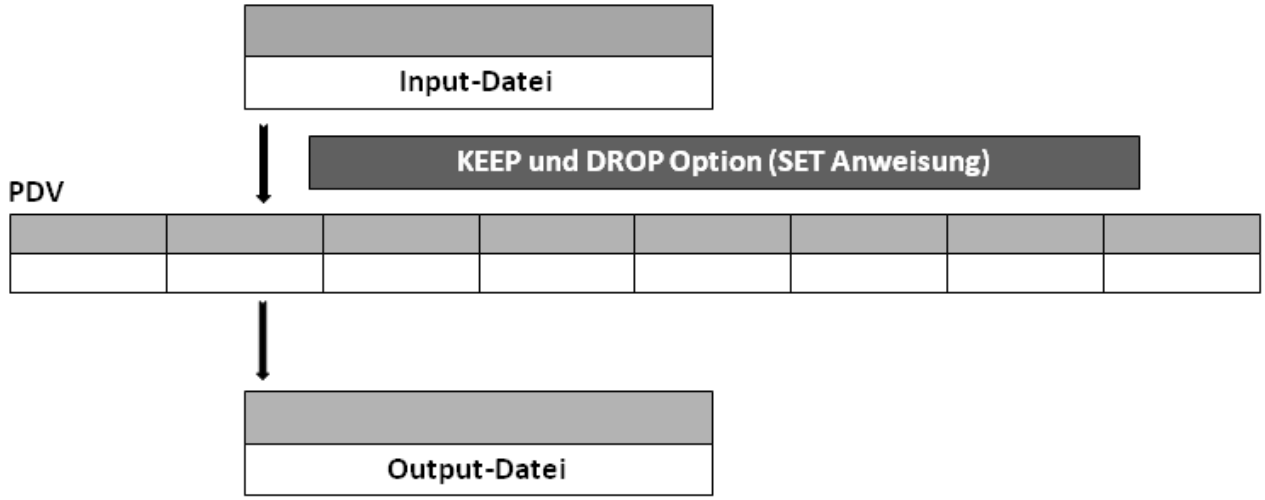

Abbildung 4: KEEP- und DROP-Option innerhalb der SET-Anweisung

Die **KEEP- und DROP-Anweisungen** sind mit den KEEP- und DROP-Optionen innerhalb der DATA-Anweisung vergleichbar. Auch hier werden alle Variablen aus der Input-Datei in den PDV eingelesen. Werden jedoch mehrere Output-Dateien innerhalb einer DATA-Anweisung definiert, die unterschiedliche Variablen beinhalten sollen, kann die Anweisung nicht verwendet werden. In diesem Fall würde zu jeder Output-Datei eine KEEP- bzw. DROP-Option gesetzt werden. Sollen mehrere Output-Dateien innerhalb eines DATA Sets erstellt werden, die alle die gleichen Variablen beinhalten sollen, so lässt sich dies mit einer einzigen KEEP- bzw. DROP-Anweisung umsetzten.

4 WHERE vs. Subsetting-IF

Mit der WHERE- bzw. Subsetting-IF-Anweisung können Beobachtungen selektiert werden, die einer bestimmten Bedingung entsprechen. Bedingungen können eine Abfolge von Operanden und Operatoren sein. Folgende Operatoren gelten sowohl für die WHERE- als auch für die Subsetting-IF-Anweisung:

Vergleichsoperatoren

```
where P1_s = 'done';
if P1_s = 'done';
```

Arithmetische Operatoren

```
where sum(P1_t,P2_t,P3_t) < 60;
if sum(P1_t,P2_t,P3_t) < 60;
```

Logische Operatoren

```
where Name = 'Müller' and P1_s = 'done';
if Name = 'Müller' and P1_s = 'done';
```

Im Folgenden soll die Verarbeitung der beiden Anweisungen grafisch gegenübergestellt werden. Abbildung 5 zeigt die Verarbeitung einer Beobachtung mit der WHERE-Anweisung. Die **WHERE-Anweisung** wird vor dem PDV ausgeführt. Bevor eine Beobachtung in den PDV eingelesen wird, wird geprüft, ob eine definierte Bedingung für diese Beobachtung erfüllt ist. Ist dies gegeben, wird die Beobachtung in den PDV geschrieben und, falls keine weiteren Anweisungen innerhalb des PDV geschaltet wurden, in die neu zu erzeugende Datei transferiert. Ist die Bedingung nicht erfüllt, wird die Beobachtung nicht in den PDV eingelesen.

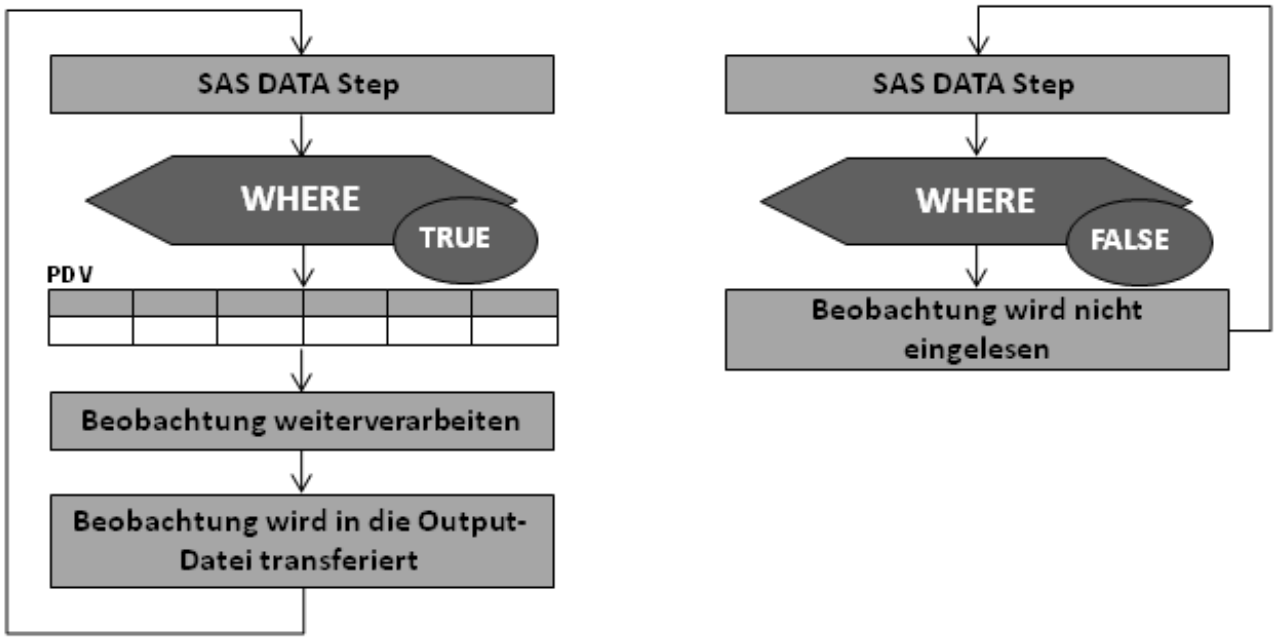

Abbildung 5: Selektion der Beobachtungen anhand der WHERE-Anweisung

In Abbildung 6 wird die Hintergrundverarbeitung des **Subsetting-IF** dargestellt. Im Gegensatz zur WHERE-Anweisung wird beim Subsetting-IF jede Beobachtung aus der Input-Datei in den PDV eingelesen. Innerhalb des PDV geschieht die Prüfung, ob die IF-Bedingung erfüllt ist oder nicht. Ist dies der Fall, wird die Beobachtung weiterverarbeitet und in die neue Datei transferiert. Ist die Bedingung nicht erfüllt, wird die Beobachtung nicht weiter verarbeitet und nicht in die Output-Datei geschrieben.

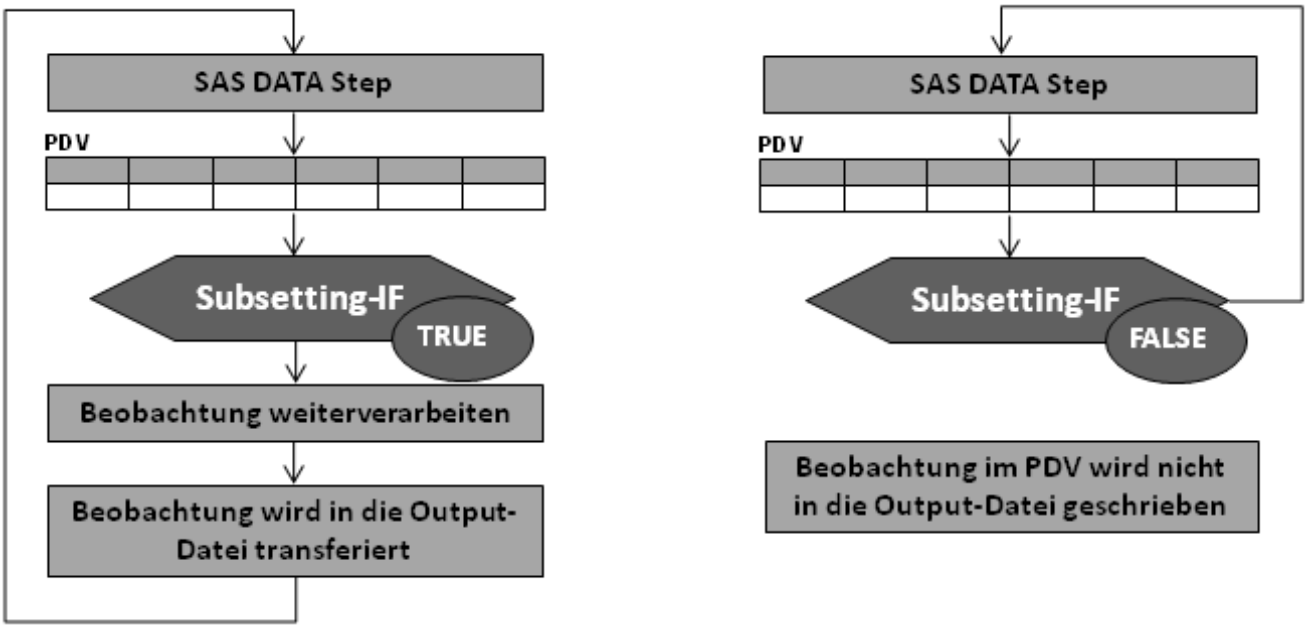

Abbildung 6: Selektion und Verarbeitung der Beobachtungen mit Subsetting-IF

Vergleicht man die WHERE- und die Subsetting-IF-Anweisungen, so lassen sich wichtige Unterschiede erkennen (siehe Tabelle 1).

Tabelle 1: WHERE vs. Subsetting-IF [4]

	WHERE	**Subsetting-IF**
Verwendung	Gültig im Daten- und Prozedurschritt (DATA Step, PROC Step)	Gültig nur im Datenschritt (DATA Step)
Zeitpunkt der Ausführung	Vor dem Einlesen einer Beobachtung in den PDV (schneller)	Nach dem Einlesen einer Beobachtung in den PDV (langsamer, da jede Beobachtung eingelesen wird)
Mehrfache Verwendung	Bei mehreren WHERE-Statements nacheinander (z.B. `where x=1; where y=5;`) wird immer nur das letzte Statement ausgeführt	Beliebig viele Subsetting-IF-Anweisungen werden einzeln ausgeführt
Temporäre Variablen	_N_, FIRST., LAST. nicht erlaubt, da diese erst im PDV bekannt sind, nachdem die WHERE-Anweisung ausgeführt wurde	Temporäre Variablen in Abfragen erlaubt
Besonderheiten	Operatoren: BETWEEN-AND, IS-NULL, IS-MISSING, CONTAINS, LIKE	

5 RETAIN-Anweisung

Variablen, die mit einer Zuweisung erzeugt werden, werden am Anfang jeder Iteration (s. Kapitel 1.2) des DATA Steps reinitialisiert. Die **RETAIN-Anweisung** verhindert die Reinitialisierung einer neu zu erzeugenden Variable während der Ausführungsphase. RETAIN wird während der Kompilierungsphase ausgeführt und die darin gesetzte Variable „gekennzeichnet". Der Wert dieser Variable wird somit gehalten und in den nächsten Datenschritt mitgeführt. Anhand des folgenden DATA Steps soll die Hintergrundverarbeitung der RETAIN-Anweisung erläutert werden.

work.Projekte

	Name	P1_t	P2_t	P3_t	P1_S	P2_S	P3_S
1	Back	21	.	25	done		done
2	Bayer	16	.	26	done		done
3	Becker	.	25	22		done	done
4	Dieter	.	27	22		tbd	tbd
5	Faust	.	25	23		done	tbd
6	Franke	14	.	22	tbd		tbd
7	Mayer	.	25	23		tbd	tbd
8	Müller	17	.	27	tbd		tbd
9	Sauer	12	.	27	tbd		tbd
10	Schmidt	15	.	22	tbd		tbd

```
data Projekt1;
  set Projekte;
  sumP_t=sum(P1_t,P2_t,P3_t);
  retain kumP1 0;
  kumP1=sum(kumP1,P1_t);
run;
```

Aus der Input-Datei **work.Projekte** soll die SAS-Datei **work.Projekt1** erzeugt werden. Dabei wird, wie auch in den vorherigen Beispielen, zum einen die Summe aller Projektzeiten berechnet (**sumP_t**) und zusätzlich die Variable **kumP1** erzeugt. **kumP1** soll die kumulierte Summe der Variable **P1_t** über die Spalte beinhalten.

Nach der Kompilierungsphase ist der PDV erzeugt und die Variablen aus der Input-Datei **work.Projekte** sowie **sumP_t** und **kumP1** angelegt. Variable **kumP1** wird dabei als eine RETAIN Variable gekennzeichnet und erhält den Initialwert 0. Mit der RETAIN-Anweisung lässt sich jeder beliebige Initialwert (Startwert im PDV) festsetzen. Wird kein Initialwert angegeben, gelten alle Werte von **kumP1** als Missing.

PDV

Name $8	P1_t N 8	P2_t N 8	P3_t N 8	P1_S $8	P2_S $8	P3_S $8	sumP_t N 8	kumP1 R N 8	_N_ N 8	_ERROR_ N 8
								0	1	0

Im ersten Schritt der Ausführungsphase wird der PDV initialisiert. Alle Variablen, außer **kumP1**, werden auf Missing gesetzt.

PDV

Name $8	P1_t N8	P2_t N8	P3_t N8	P1_S $8	P2_S $8	P3_S $8	sumP_t N8	kumP1 N8	_N_ N8	_ERROR_ N8
	.	.	.				.	0	1	0

Nacheinander werden die Werte der ersten Beobachtung aus der Input-Datei in den PDV geschrieben, anschließend der Wert der Variable **`sumP_t`** berechnet und zuletzt die erste Berechnung für **`kumP1`** durchgeführt:

PDV

0 + 21

Name $8	P1_t N8	P2_t N8	P3_t N8	P1_S $8	P2_S $8	P3_S $8	sumP_t N8	kumP1 N8	_N_ N8	_ERROR_ N8
Back	21	.	25	done		done	46	21	1	0

Die Beobachtung wird in die Output-Datei **`work.Projekt1`** transferiert:

`work.Projekt1`

Name	P1_t	P2_t	P3_t	P1_S	P2_S	P3_S	sumP_t	kumP1
Back	21	.	25	done		done	46	21

Am Anfang des nächsten Datenschrittes erfolgt die Reinitialisierung des PDV. Hierbei wird nur **`sumP_t`** auf Missing gesetzt. Die Variablen, die aus der SAS-Datei eingelesen wurden, sowie die RETAIN Variable, behalten den vorherigen Wert.

PDV

Name $8	P1_t N8	P2_t N8	P3_t N8	P1_S $8	P2_S $8	P3_S $8	sumP_t N8	kumP1 N8	_N_ N8	_ERROR_ N8
Back	21	.	25	done		done	.	21	2	0

Die Werte der zweiten Beobachtung werden eingelesen:

PDV

21 + 16

Name $8	P1_t N8	P2_t N8	P3_t N8	P1_S $8	P2_S $8	P3_S $8	sumP_t N8	kumP1 N8	_N_ N8	_ERROR_ N8
Bayer	16	.	26	done		done	42	37	2	0

Die zweite Beobachtung wird in **`work.Projekt1`** transferiert.

`work.Projekt1`

Name	P1_t	P2_t	P3_t	P1_S	P2_S	P3_S	sumP_t	kumP1
Back	21	.	25	done		done	46	21
Bayer	16	.	26	done		done	42	37

Analog werden alle weiteren Beobachtungen bis zum End of File ausgeführt.

Variablen, die innerhalb eines DATA Steps neu gebildet werden, werden grundsätzlich am Ende der neuen SAS-Datei gesetzt. Sie werden chronologisch in der Reihenfolge

ihrer Entstehung und nicht alphabetisch vom System geordnet [5]. Möchte man eine neu erzeugte Variable am Anfang der Tabelle setzen, eignet sich hierfür die RETAIN-Anweisung.

```
data Projektzeit_Mitarbeiter;
  retain Name sumP_t;
  set Projekte;
  sumP_t=sum(P1_t,P2_t,P3_t);
run;
```

	Name	sumP_t	P1_t	P2_t	P3_t	P1_S	P2_S	P3_S
1	Back	46	21	.	25	done		done
2	Bayer	42	16	.	26	done		done
3	Becker	47	.	25	22		done	done
4	Dieter	49	.	27	22		tbd	tbd
5	Faust	48	.	25	23		done	tbd
6	Franke	36	14	.	22	tbd		tbd
7	Mayer	48	.	25	23		tbd	tbd
8	Müller	44	17	.	27	tbd		tbd
9	Sauer	39	12	.	27	tbd		tbd
10	Schmidt	37	15	.	22	tbd		tbd

Allerdings muss die RETAIN-Anweisung im DATA Step vor der SET-Anweisung gesetzt werden, da sonst die Reihenfolge der Variablen durch die SET-Anweisung definiert wird.

6 Schlussfolgerung

Das Verständnis der Hintergrundprozesse eines SAS DATA Steps kann die Performance der Datenverarbeitung erheblich steigern. Bei der Verarbeitung großer Datenmengen ist es von Vorteil, die verschiedenen Optionen und Anweisungen bestmöglich zu verwenden und damit effiziente DATA Steps und so eine schnelle und fehlerfreie Verarbeitung zu gewährleisten.

Literatur

[1] SAS Help: Starting with Raw Data: The Basics

[2] Gigic B., Deckert A.: SAS Backstage. KSFE 2012 - Proceedings der 16. Konferenz der SAS®-Anwender in Forschung und Entwicklung. Shaker Verlag; 103-111

[3] SAS Institute Inc.: SAS® Certification Prep Guide: Base Programming for SAS®, Third Edition. Cary, NC: SAS Institute Inc., 2011

[4] http://www.urz.uni-heidelberg.de/statistik/sas-ah/01.03/selektion.html [21.03.2013]

[5] http://www.urz.uni-heidelberg.de/statistik/sas-ah/05.01/SAS-Kurs.html [21.03.2013]