Performancetuning für SAS Programme

Andreas Bachert
HMS Analytical Software
Rohrbachert Straße 26
69115 Heidelberg
Andreas.bachert@analytical-sofware.de

Zusammenfassung

Oft ist es nicht auf den ersten Blick ersichtlich, warum ein SAS-Programm viel Rechenzeit benötigt. Dieser Vortrag stellt verschiedene Techniken für die Optimierung der Laufzeit von SAS-Programmen bei der Verarbeitung großer Datenmengen vor. Dabei wird auch auf die Hintergründe der resultierenden Performanceunterschiede eingegangen.

Zu den anhand von Beispielprogrammen vorgestellten Techniken zählen:

- SAS-Systemoptionen mit Auswirkung auf die Performance
- Allgemeine laufzeitoptimierende Programmiertechniken
- Arbeiten mit Formaten
- Arbeiten mit Views und Indizes
- Neue Lookup-Techniken
- Laden von Datasets in den Hauptspeicher

Die Beispiele basieren auf SAS in der Version 9.2.

Schlüsselwörter: Compress, Remerge, Sort, NoEquals, Format, Index, View, SASFile, HashTable, Performance

1 Allgemeine Betrachtungen zu Performance

Wer einfach nur mit SAS oder dem SAS Enterprise Guide (SEG) Datenaufbereitungs-, Analyse- oder Berichtsprogramme entwickelt und dabei über genug Zeit und Plattenplatz verfügt, braucht sich letztlich nicht mit dem Thema Programm-Performance auseinandersetzen.
Nur wer unter zu langen Laufzeiten leidet, bzw. wer darauf achten muss, dass Datasets nicht zu groß werden, muss hier tätig werden.
Oft sind es Batch-Läufe, wo die Gefahr besteht, dass Anwender bereits auf neue Daten zugreifen wollen, bevor die Ladeprogramme überhaupt zu Ende gelaufen sind.
Desweiteren ist es wichtig, zu erreichen, dass Berichtsprogramme möglichst schnell sind. Wer will denn schon 30 Sekunden auf einen Bericht warten, wenn er evtl. auch in 2 Sekunden fertig sein könnte?
Vielleicht ist es auch so, dass ein Programm, das sich schon im produktiven Einsatz befindet, mehr und mehr Kummer auslöst. Schuld daran ist dann meistens zunehmendes Datenvolumen, das verarbeitet werden muss. Der Ansatz, der dann verfolgt werden

muss, ist die Laufzeit wenigstens gleich zu halten und/oder den Plattenplatz zu verringern.

In diesem Beitrag werden unterschiedliche Szenarien vorgestellt, wobei jeweils Vorschläge gemacht werden, wie man in diesen Fällen mit einfachen Mitteln die Performance positiv beeinflussen kann.
Abschließend werden zusammenfassend einige Tipps gegeben, die man immer berücksichtigen kann, da sie nichts kosten und immer Vorteile bringen.

1.1 Was kann man grundsätzlich tun?

Ein adäquates Mittel um Performance zu steigern ist immer die Hardware auszubauen:

- Einbau zusätzlicher Festplatten
- Verteilung von Daten auf verschiedene Festplatten
    - Dadurch kann Parallelisierung bei Plattenzugriffen erreicht werden
- Bereitstellung von mehr Hauptspeicher
- Bereitstellung von zusätzlichen Prozessoren
- Erhöhung der Netzwerkbandbreite

Sollte hardwareseitig das Optimum bzw. das Machbare erreicht worden sein, muss man sich spätestens darum kümmern, welche Möglichkeiten das SAS System in diesem Zusammenhang bietet. Diese sollten dann ausgeschöpft werden.

Zu guter letzt kann man sich um die Programme selbst kümmern. Hier finden sich in der Regel Ansätze, um die Performance zu verbessern.

1.2 Möglichkeiten, zur Ressourcenkontrolle in SAS

Mit Hilfe der folgenden beiden Systemoptionen kann man sich Informationen über den Ressourcenverbrauch von SAS Programmen verschaffen.

- FullSTimer
    - Liefert Informationen über die Dauer, die verbrauchte CPU-Zeit und den Hauptspeicherverbrauch eines DATA oder PROC Steps
- SASTrace
    - Gibt Aufschluss darüber, wie der Zugriff auf relationale Datenbanksysteme (RDBMS) von der entsprechenden SAS ACCESS Engine durchgeführt wurde.
    - Welche Teile wurden direkt an die Datenbank weitergeleitet und welche Teile wurden von SAS ausgeführt?

Sehr nützlich ist es bei Performancemessungen auch, sich selbst zwei SAS Macros zu schreiben, mit denen eine Echtzeitmessung durchgeführt werden kann. Das erste Macro initialisiert die Zeitmessung und das zweite berechnet die Zeit, die seit der Initialisierung vergangen ist und speichert diese Information in globalen Macrovariablen oder in einer Tabelle.

1.3 SAS Optionen zur Ressourcensteuerung

Neben den Optionen zur Kontrolle von Ressourcenverbräuchen gibt es im SAS System zahlreiche System Optionen, mit denen die Verwendung der Ressourcen gesteuert werden kann.

- CPUCount
  - Anzahl der Prozessoren, die SAS nutzen darf
- MemSize
  - Max. Hauptspeicher
- SortSize
  - Hauptspeicherfestlegung für Sortierschritte
- SumSize
  - Hauptspeicherfestlegung für Aggregationen
- BufSize
  - Größe eines Datenpaketes in einem I/O Step
- BufNo
  - Anzahl der Datenpakete
- Threads
  - Sollen SAS-Steps in Threads aufgeteilt werden?

1.4 Tipps zu Performancemessungen

Folgende Punkte sollten beachtet werden, wenn Performancemessungen durchgeführt werden.

- Einschalten der Optionen zur Performance Messung
- Jede Variante/Technik in eigener Session testen
- Für jede Messung nur eine Technik ändern, damit die gemessenen Unterschiede richtig beurteilt werden können
- Tests unter möglichst produktionsnahen Bedingungen ausführen
- Jeden Tests mehrmals durchführen und Mittelwerte bestimmen

2 Konkrete Programmierszenarien

Im Folgenden werden verschiedene Szenarien kurz umrissen, wobei jeweils eine Empfehlung gegeben wird, wie im jeweiligen Fall verfahren werden soll. Zudem ist ein Beispielprogramm angegeben, das die Umsetzung veranschaulicht.

2.1 Große Datasets mit vielen Alphanumerischen Variablen

- Szenario
  - Große Dateien mit vielen CHAR-Variablen
- Empfehlung
  - SAS Option COMPRESS=CHAR einschalten und in Ausnahmefällen die Dataset-Option COMPRESS=NO für einen Zugriff aktivieren

- Vorteile
  - Spart Plattenplatz
  - Reduziert I/O
  - SAS entscheidet selbst, wann sich komprimieren lohnt
- Nachteile
  - Höhere CPU-Auslastung
  - Aber: CPU immer schneller als I/O, so dass am Ende neben Plattenplatzgewinn auch Zeitgewinn steht ☺
- Beispielprogramm

```
/*************************************************************/
/* -  Beispiel mit COMPRESS=NO
      -  DATA STEP
         -  Lesen aus einer Datei mit vielen großen Textspalten
         -  Schreiben einer Kopie
      -  DATA STEP
         -  Lesen aller Kunden aus der Kopie, deren Nachname
            NICHT mit A beginnt
         -  Speichern des Subsets
*/
OPTIONS
   COMPRESS = NO
;
DATA Daten.PotentialUnCompressed;
   SET Daten.Potential_Customers_Large;
RUN;
DATA Daten.PotentialUnCompressed_Not_A;
   SET   Daten.PotentialUnCompressed;
   WHERE (Substr (Customer_LastName, 1, 1) NE "A");
RUN;

/*************************************************************/
/* -  Gleiches Beispiel mit COMPRESS=YES als Dataset-Option
      -  Das Lesen der unkomprimierten Datei geht langsam, das
         Schreiben
         mit COMPRESS=YES geht schon schnell
      -  Besser noch ist es, die System-Option COMPRESS=YES zu
         setzen
         und nur im Ausnahmefall die Dataset-Option COMPRESS=NO
         zu
         verwenden
*/
DATA Daten.PotentialCompressed (Compress=CHAR);
   SET Daten.Potential_Customers_Large;
RUN;
OPTIONS
   COMPRESS = YES
;
DATA Daten.PotentialCompressed_Not_A;
   SET   Daten.PotentialCompressed;
```

```
   WHERE (Substr (Customer_LastName, 1, 1) NE "A");
RUN;
```

2.2 Aggregieren mit PROC SQL – Versehentliches remergen

- Szenario
  - Bei Verwendung der Aggregationsfunktionen in PROC SQL führt falsches Gruppieren zu wiederholtem Lesen der Daten (Remerging)
- Empfehlung
  - Im SELECT Statement zuerst alle Variablen, dann alle Aggregationsfunktionen aufführen
  - Im GROUP BY Statement alle Variablen aus dem SELECT in der gleichen Reihenfolge wiederholen
  - SAS Option SQLNOREMERGE einschalten und in Ausnahmefällen die Option REMERGE für einen PROC SQL-Schritt aktivieren
- Vorteile
  - Versehentliches "Remergen" wird verhindert
- Nachteile
  - Keine
- Beispielprogramm

```
OPTIONS
   SQLREMERGE
;

/*************************************************************/
/* -  Beispiel mit unkorrekter Gruppierung
      -  An die Kunden-ID werden Name usw. aus anderer Tabelle
         angespielt
      -  Die Gruppierung erfolgt aber nur über Kunden-ID
      -  SAS muss zuerst Summe je Kunde ermitteln und diese
         danach
         an jeden Datensatz anhängen
      => Die Ursprungstabelle muss 2 mal komplett durchlaufen
         werden
*/
TITLE    "Summe der Umsätze je Kunde";
TITLE2   "Gruppierung nur über Kunden-ID!!";
TITLE3   "-> Kunden kommen sooft vor wie es Aufträge pro Kunde
gibt";
PROC SQL NOPRINT;
   CREATE TABLE RevenuePerCustomerDuplicates AS
      SELECT
               B.Customer_ID
            ,  B.Customer_FirstName
            ,  B.Customer_LastName
            ,  Sum (A.Total_Retail_Price) AS Customer_Revenue
      FROM     Daten.Order_Fact     A
            ,  Daten.Customer_Dim   B
```

```
      WHERE    A.Customer_ID  EQ B.Customer_ID
      GROUP BY B.Customer_ID
   ;
QUIT;
PROC PRINT
   DATA  = &SYSLast. (OBS = 20);
RUN;



/*************************************************************/
/* -  Beispiel mit unkorrekter Gruppierung, wobei jedoch
      Zusätzlich SELECT DISTINCT verwendet wird
      => SAS muss immer noch 2 mal durch die Daten laufen,
         wobei jetzt zusätzlich das Ergebnis noch
         eindeutig gemacht werden muss!!
*/
TITLE    "Summe der Umsätze je Kunde";
TITLE2   "Gruppierung nur über Kunden-ID, jedoch SELECT
DISTINCT!!";
TITLE3   "-> Ergebnis ist korrekt, aber noch mehr
Performanceverlust";
PROC SQL NOPRINT;
   CREATE TABLE RevenuePerCustomerDistinct AS
      SELECT   DISTINCT
               B.Customer_ID
             , B.Customer_FirstName
             , B.Customer_LastName
             , Sum (A.Total_Retail_Price) AS Customer_Revenue
      FROM     Daten.Order_Fact     A
             , Daten.Customer_Dim   B
      WHERE    A.Customer_ID  EQ B.Customer_ID
      GROUP BY B.Customer_ID
   ;
QUIT;
PROC PRINT
   DATA  = &SYSLast. (OBS = 20);
RUN;



/*************************************************************/
/* -  Beispiel mit korrekter Gruppierung über alle Variablen,
      die ohne Aggregations-Funktion im SLECT Statement
      aufgeführt sind
      => Kein REMERGE mehr notwendig
*/
TITLE    "Summe der Umsätze je Kunde";
TITLE2   "Richtig gruppiert, schon ist's passiert";
PROC SQL NOPRINT;
   CREATE TABLE RevenuePerCustomer AS
      SELECT   B.Customer_ID
```

```
            ,  B.Customer_FirstName
            ,  B.Customer_LastName
            ,  Sum (A.Total_Retail_Price) AS Customer_Revenue
      FROM     Daten.Order_Fact     A
            ,  Daten.Customer_Dim   B
      WHERE    A.Customer_ID  EQ B.Customer_ID
      GROUP BY B.Customer_ID
            ,  B.Customer_FirstName
            ,  B.Customer_LastName
   ;
QUIT;
PROC PRINT
   DATA  = &SYSLast. (OBS = 20);
RUN;



/*************************************************************/
/* -  Ungewollt falsches Gruppieren kann mit der SAS Option
      NOSQLREMERGE verhindert werden
*/
TITLE "Falsche Gruppierung per OPTION verhindern";
OPTIONS
   NOSQLREMERGE
;
PROC SQL NOPRINT;
   CREATE TABLE RevenuePerCustomerError AS
      SELECT   DISTINCT
               B.Customer_ID
            ,  B.Customer_FirstName
            ,  B.Customer_LastName
            ,  Sum (A.Total_Retail_Price) AS Customer_Revenue
      FROM     Daten.Order_Fact     A
            ,  Daten.Customer_Dim   B
      WHERE    A.Customer_ID  EQ B.Customer_ID
      GROUP BY B.Customer_ID
   ;
QUIT;



/*************************************************************/
/* -  Beispiel, wo das Remergen bei einem Join sinnvoll ist
      -  Es soll der Prozentsatz des Umsatzes jedes einzelnen
         Kunden am Gesamtumsatz berechnet werden
*/
TITLE    "Anteil jedes einzelnen Kunden am Gesamtumsatz";
TITLE2   "Beispiel für sinnvollen Einsatz von REMERGE";
TITLE3   "Dazu wurde im PROC SQL die Option REMERGE aktiviert";
PROC SQL NOPRINT REMERGE;
   CREATE TABLE RevenuePerCustomerPct AS
      SELECT   Customer_ID
```

```
            ,  Customer_FirstName
            ,  Customer_LastName
            ,  Customer_Revenue          Format=DollarX12.2
            ,  Sum (Customer_Revenue)
                  AS Overall_Revenue     Format=DollarX12.2
            ,  Customer_Revenue / CALCULATED Overall_Revenue
                  AS Percentage          Format=Percent8.2
      FROM     RevenuePerCustomer
   ;
QUIT;
PROC PRINT
   DATA  = &SYSLast. (OBS = 20);
RUN;
```

2.3 Daten effizient sortieren

- Szenario
  - Sortieren von Daten mit PROC SORT
- Empfehlung
  - SAS Option NOSORTEQUALS einschalten und in Ausnahmefällen die Option EQUALS für einen PROC SORT-Schritt aktivieren
- Vorteile
  - Weniger Ressourcenverbrauch beim Sortieren und somit hoher Performancegewinn
- Nachteile
  - Datensätze mit gleichen Ausprägungen in den Sortiervariablen sind im Ergebnis in zufälliger Reihenfolge angeordnet
  - Aber: Vorherige Reihenfolge der Daten ist meistens egal ☺
- Beispielprogramm

```
/*************************************************************/
/* -  Normales Sortieren mit der SAS System Optioneinstellung
      SORTEQUALS
*/
OPTIONS
   SORTEQUALS
;
TITLE    "Daten wurden mit EQUALS nach Street sortiert";
PROC SORT
   DATA  = AddressesPropCase
   OUT   = AddressesEqual;
   BY    Street;
RUN;
PROC PRINT
   DATA  = &SYSLast. (OBS = 20);
RUN;



/*************************************************************/
```

```
/* -  Beispiel mit PROC SORT und NOEQUALS
      -  Reihenfolge bei gleicher Straße ist 'zufällig'
      -  Besser noch ist es, die System-Option NOSORTEQUALS zu
         Setzen und nur im Ausnahmefall die PROC SORT Option
         EQUALS zu verwenden
*/
TITLE    "Daten wurden mit NOEQUALS nach Street sortiert";
TITLE2   "Andere Reihenfolge als nach dem Sortieren mit EQUALS
bei gleicher Straße beachten!!";
PROC SORT NoEquals
   DATA  = AddressesPropCase
   OUT   = AddressesNoEqual;
   BY    Street;
RUN;
PROC PRINT
   DATA  = &SYSLast. (OBS = 20);
RUN;

OPTIONS
   NOSORTEQUALS
;
```

2.4 Gruppieren mittels Formaten

- Szenario
  - Daten sollen gruppiert ausgewertet werden
  - Gruppierung ist nicht in den Daten gespeichert
  - Originalreihenfolge der Daten soll beibehalten werden
- Empfehlung
  - Eigenes Format mit Gruppierungsinformation erstellen und in den Prozeduren verwenden
- Vorteile
  - Zusätzliche Schritte zur Erzeugung und Speicherung der Gruppierungsinformation in neuen Variablen entfallen
  - Sortierreihenfolge ist wählbar (unformatiert / formatiert)
  - Formatierte Variablen können CLASS und VAR sein
- Nachteile
  - Zusätzlicher Schritt für Formaterzeugung ist notwendig
  - Aber: Man kann permanente Formate verwenden ☺
- Beispielprogramm

```
/*************************************************************/
/* SZENARIO:
   ========
   -  Kunden sollen in Abhängigkeit des Umsatzes in 4 Gruppen
      eingeteilt werden.
   -  Danach ist eine Auswertung je Gruppe gefordert mit
      Standardstatistiken
      -  Summe
```

```
      -  Mittelwert
      -  ...
   -  In diesem Beispiel wird zuerst eine neue Variable
      'RevenueGroup' erzeugt und
      das entstehende Ergebnis wird als WORK-Datei gespeichert
   -  Ein PROC MEANS wertet dann die WORK-Datei aus
*/
TITLE "Gruppierung mittels separater Variable";
TITLE2 "Sortierreihenfolge entspricht nicht der
Datenreihenfolge!";
DATA RevenueGroups;
   SET RevenuePerCustomer;
   LENGTH
      RevenueGroup      $20
   ;
   SELECT;
      WHEN (Customer_Revenue LT 1000) DO;
         RevenueGroup   = 'Unter 1000';
      END;

      WHEN ((Customer_Revenue GE 1000)
             AND (Customer_Revenue LT 3000)) DO;
         RevenueGroup   = '1000 bis 3000';
      END;

      WHEN ((Customer_Revenue GE 3000)
             AND (Customer_Revenue LT 5000)) DO;
         RevenueGroup   = '3000 bis 5000';
      END;

      OTHERWISE DO;
         RevenueGroup   = 'Über 5000';
      END;
   END;
RUN;
PROC MEANS
   DATA = RevenueGroups;
   CLASS    RevenueGroup;
   VAR      Customer_Revenue;
RUN;



/*************************************************************/
/* -  In diesem Beispiel wird zunächst ein Format erzeugt, das
      die Gruppen definiert
   -  Im PROC MEANS wird dann die Umsatz-Variable im CLASS
      Statement und im VAR Statement angegeben, wobei zudem die
      Verwendung des Gruppierungsformates bestimmt wird
   -  Auch in einem PROC GCHART kann die Variable unter
      Verwendung des Formates als Diskrete Variable ausgewertet
      werden
```

```
*/
TITLE  "Gruppierung über formatierte Werte";
TITLE2 "Sortierreihenfolge entspricht der Datenreihenfolge!";
TITLE3 "Formatierte Variable kann CLASS und VAR sein!";
PROC FORMAT;
   VALUE CustomerClass
      LOW   -  1000  = 'Unter 1000'
      1000  -  3000  = '1000 bis 3000'
      3000  -  5000  = '3000 bis 5000'
      OTHER          = 'Über 5000';
RUN;
PROC MEANS
   DATA = RevenuePerCustomer;
   CLASS    Customer_Revenue;
   VAR      Customer_Revenue;
   FORMAT   Customer_Revenue CustomerClass.;
RUN;
TITLE    "Häufigkeitsverteilung über formatierte Werte";
TITLE2   "Die Gruppierung ergibt sich aus der Verwendung des
Formats";
PROC GCHART
   DATA = RevenuePerCustomer;
   VBAR3D Customer_Revenue / DISCRETE;
   FORMAT   Customer_Revenue CustomerClass.;
RUN;QUIT;

/************************************************************
*********/
TITLE;
```

2.5 Filtern mit Hilfe von Indizes

- Szenario
  - Häufige Selektionen mit Abfrage von Variablen mit hoher Selektivität
  - Daten ändern sich selten
- Empfehlung
  - Daten Indexieren
  - SAS Option MSGLEVEL=I einschalten
- Vorteile
  - Performancegewinn bei SQL Join und DATA STEP Merge
  - Ermöglicht BY-Verarbeitung ohne vorheriges Sortieren
  - Ermöglicht die Verwendung von KEY= und _IORC_ im DATA STEP
- Nachteile
  - Zusätzliche Index-Datei auf Platte
  - Pflege des Index ist aufwändig und kostet Zeit. Oft ist das Neuschreiben schneller als Datensätze einzufügen
- Beispielprogramm

```
OPTIONS
```

```
   MSGLEVEL = I
;
/*************************************************************
   -  Es wird häufig über die ID und den Namen der Kunden auf
      eine Kundendatei
      zugegriffen
   -  Erzeugen von Indices via DATA STEP
      -  Alternative Möglichkeiten:
         -  PROC SQL
         -  PROC DATASETS
*/
DATA Potential_Customers   (Index =
                      (Customer_ID       = (Customer_ID)
                       Customer_LastName = (Customer_LastName)
                      )
                            );
   SET Daten.Potential_Customers;
RUN;


/*************************************************************
   -  Abfrage auf Gleichheit führt zur Indexverwendung
*/
DATA PotentialCustomers_Subset_Klem;
   SET   Potential_Customers;
   WHERE (Customer_LastName EQ "Klem");
RUN;


/*************************************************************
   -  Diese Abfrage ist noch selektiv genug,
      so dass Index verwendet wird
*/
DATA PotentialCustomers_Subset_Kl;
   SET   Potential_Customers;
   WHERE (Customer_LastName LIKE "Kl%");
RUN;


/*************************************************************
   -  Abfrage zu ungenau, so dass Index nicht helfen kann
   -  Hinweis im Log beachten, dass das Sortieren helfen kann
*/
DATA PotentialCustomers_Subset_K;
   SET   Potential_Customers;
   WHERE (Customer_LastName LIKE "K%");
RUN;


/*************************************************************
   -  Wenn die Daten vor der Indexierung nach der
Selektionsvariable sortiert
```

```
      werden, hilft das bei der späteren Selektion
   -  Der Index wird nun doch verwendet
*/
PROC SORT FORCE
   DATA  = Daten.Potential_Customers
   OUT   = Potential_Customers;
   BY       Customer_LastName;
RUN;

DATA Potential_Customers   (Index =
                      (Customer_ID       = (Customer_ID)
                       Customer_LastName = (Customer_LastName)
                      )
                           );
   SET Potential_Customers;
RUN;
DATA PotentialCustomers_Subset_K;
   SET   Potential_Customers;
   WHERE (Customer_LastName LIKE "K%");
RUN;
```

2.6 Views statt Zwischentabellen

- Szenario
  - Ursprungsdaten werden in aufeinanderfolgenden Schritten modifiziert und es entstehen Zwischenergebnisse
- Empfehlung
  - Immer die Verwendung von Views prüfen
  - Zu Testzwecken ggf. kurzzeitig Datasets erzeugen
- Vorteile
  - Eine VIEW benötigt praktisch keinen Speicherplatz
  - Bei Ausführung findet die Verarbeitung im Hauptspeicher und via schnellen temporären Tabellen statt
- Nachteile
  - Bei jedem Zugriff auf die VIEW werden die Ursprungsdaten erneut gelesen und ggf. komplex verarbeitet
- Beispielprogramm

```
/*************************************************************/
/*
-  Es sollen die fortlaufenden kumulativen Umsätze aus den
   Aufträgen des Jahres 2004
   ausgegeben werden
-  Beispiel 1:
   -  PROC SORT:   -  Sortieren der Augangsdaten nach Datum.
                   -  Speichern des Sortierergebnisses als
                      Zwischentabelle
   -  DATA STEP:   -  Berechnen des kumulierten Ergebnisses je
                      Datensatz.
```

```
                        Dabei erzeugen der Variablen 'OrderYear'
                        und 'OrderMonth'
                    -   Speichern des Ergebnisses als
                        Zwischentabelle
    -   DATA STEP:  -   Berechnen des kumulierten Ergebnisses je
                        Jahr.
                    -   Speichern des Ergebnisses als
                        Zwischentabelle
*/
TITLE  "Kumulative Umsätze für das Jahr 2004";
TITLE2 "Beispiel 1: Drei Zwischentabellen mit allen Variablen";
%TimeMeasurementInit;
PROC SORT NOEQUALS
   DATA  = Daten.Order_Fact
   OUT   = Order_Fact;
   BY       Order_Date;
RUN;
DATA CumulativeRevenueTotal;
   SET Order_Fact;
   LENGTH
      OrderYear          8
      OrderMonth         $7
   ;
   RETAIN
      CumRevenueTotal    0
   ;
   OrderYear          = Year (Order_Date);
   OrderMonth         = CatS (OrderYear, '_'
                            , Put (Month (Order_Date), Z2.));
   CumRevenueTotal    = CumRevenueTotal + Total_Retail_Price;
RUN;
DATA CumulativeRevenueYear;
   SET CumulativeRevenueTotal;
   BY  OrderYear;
   RETAIN
      CumRevenueYear     0
   ;
   IF (FIRST.OrderYear) THEN DO;
      CumRevenueYear = 0;
   END;
   CumRevenueYear = CumRevenueYear + Total_Retail_Price;
RUN;
PROC PRINT
   DATA = CumulativeRevenueYear;
   VAR       Order_ID Order_Date OrderMonth
             Total_Retail_Price CumRevenueYear;
   FORMAT    CumRevenueYear  DollarX12.2;
   WHERE     OrderYear EQ 2004;
RUN;
%TimeMeasurementStop;
```

```
/*************************************************************/
/*
-  Beispiel 2:
   -  PROC SQL:    -  Definieren einer View, die nur die
                      notwendigen Variablen enthält
                      und die neuen Variablen 'OderYear' und
                     'OrderMonth' enthält
                   -  Speichern als SQL-View
   -  DATA STEP:   -  Berechnen der kumulierten Ergebnisse auf
                      Gesamt- und auf Jahresebene.
                   -  Speichern als DATA STEP-View
*/
TITLE2 "Beispiel 2: Gleiches Ergebnis mit SQL View und Datastep
View";
TITLE3 "Es werden nur notwendige Variablen behalten";
%TimeMeasurementInit;
PROC SQL NOPRINT;
   CREATE VIEW vwOrderFact AS
      SELECT   Order_ID
             ,  Order_Date
             ,  Year (Order_Date)  AS OrderYear
             ,  CatS (CALCULATED OrderYear, '_'
                        , Put (Month (Order_Date), Z2.))
                                   AS OrderMonth
             ,  Total_Retail_Price
      FROM     Daten.Order_Fact
      ORDER BY Order_Date
   ;
QUIT;
DATA CumulativeRevenue / View=CumulativeRevenue;
   SET vwOrderFact;
   BY  OrderYear OrderMonth;
   RETAIN
      CumRevenueTotal    0
      CumRevenueYear     0
   ;
   IF (FIRST.OrderYear) THEN DO;
      CumRevenueYear = 0;
   END;
   CumRevenueTotal    = CumRevenueTotal + Total_Retail_Price;
   CumRevenueYear     = CumRevenueYear + Total_Retail_Price;
RUN;
PROC PRINT
   DATA = CumulativeRevenue;
   VAR       Order_ID Order_Date OrderMonth
             Total_Retail_Price CumRevenueYear;
   FORMAT    CumRevenueYear  DollarX12.2;
   WHERE     OrderYear EQ 2004;
RUN;
TITLE;
%TimeMeasurementStop;
```

2.7 SASFILE Statement

- Szenario
  - Eine Datei wird im Gesamtablauf häufig gelesen bzw. modifiziert
  - (Letztlich Gegenbeispiel zur Verwendung von VIEWS)
- Empfehlung
  - Datei mit SASFILE in den Hauptspeicher laden
- Vorteile
  - Zugriff auf die Datei geht sehr schnell
- Nachteile
  - Nur sinnvoll bei nicht zu großen Dateien und genügend Hauptspeicher
- Beispielprogramm

```
dm WEdit 'Clear Log' WEdit;
/*************************************************************/
/*
-  Vorbereiten der Beispieldatei
*/
PROC SQL NOPRINT;
   CREATE TABLE Revenue AS
      SELECT
                B.Customer_ID
              ,  B.Customer_FirstName
              ,  B.Customer_LastName
              ,  CatX (", ", B.Customer_LastName
                           , B.Customer_FirstName)
                     AS Customer_Name
              ,  A.Order_Date
              ,  Total_Retail_Price   LABEL="Revenue"
                                      FORMAT=DollarX12.2
      FROM      Daten.Order_Fact      A
              ,  Daten.Customer_Dim    B
      WHERE     A.Customer_ID  EQ B.Customer_ID
   ;
QUIT;

/* Diese Datei laden, da sie häufig gebraucht wird */
SASFILE Revenue load;

TITLE "Kundenliste mit Umsatzanzeige (Netto)";
proc print data=Revenue (OBS = 50);
   var Customer_ID Order_Date Customer_Name Total_Retail_Price;
run;

TITLE "Auswertung der Umsätze je Kunde (Netto)";
proc tabulate data=Revenue;
   class Customer_Name;
   var   Total_Retail_Price;
   table Customer_Name, Total_Retail_Price*(n
sum*FORMAT=DollarX12.2 mean*FORMAT=DollarX12.2);
run;
```

```
/*************************************************************/
/* -  Berechnen der Brutto-Umsätze
   -  Schreiben in eine mit SASFILE geladenes Dataset ist
ebenfalls möglich
*/
DATA Revenue;
   MODIFY Revenue;
   Total_Retail_Price   = Total_Retail_Price * 1.19;
RUN;

TITLE "Kundenliste mit Umsatzanzeige (Brutto)";
proc print data=Revenue (OBS = 50);
   var Customer_ID Order_Date Customer_Name Total_Retail_Price;
run;
TITLE "Auswertung der Umsätze je Kunde (Brutto)";
proc tabulate data=Revenue;
   class Customer_Name;
   var   Total_Retail_Price;
   table Customer_Name
       , Total_Retail_Price*(n sum*FORMAT=DollarX12.2
                             mean*FORMAT=DollarX12.2);
run;
TITLE;

/* Die Datei wieder entladen und somit den Hauptspeicher
freigeben */
SASFILE Revenue close;
```

2.8 Lookup mit Hashtables

- Szenario
  - In einem DATA STEP gleichzeitig nach mehreren Informationen in anderen Tabellen nachschlagen
- Empfehlung
  - Nachschlagen über Hash-Tabellen
- Vorteile
  - Sehr schnell
  - Mehrere Hash-Tabellen gleichzeitig
  - Mehrere Schlüssel-Spalten und mehrere Daten-Spalten sind möglich
- Nachteile
  - Nur möglich im DATA STEP
  - Alles muss in den Hauptspeicher geladen werden
- Beispielprogramm

```
dm WEdit 'Clear Log' WEdit;
/*
   -  Anspielen von Informationen über Kunde, Mitarbeiter und
      Produkt mit Hashtable
   -  Vorteile von Hashtables
      -  Mehr als eine Keyspalte können definiert werden
```

```
            -  Kann mehr als ein Spalte aus der Lookuptabelle als
               Treffer zurückliefern
            -  Beteiligte Tabellen müssen nicht sortiert sein
            -  Es können mehrere Hash-Objekte angelegt werden
            -  Hashtable wird nur einmal zu Beginn des Datasteps
               gelesen und die Info befindet sich dann im
               Hauptspeicher
               -  Sehr schnell
      */
      data  Order_Fact_Complete
            Unknown_Customer
            Unknown_Employee
            Unknown_Product
            ;

         Set Daten.Order_Fact;

         if (_N_ EQ 1) then do;
            if (1 EQ 2) then do;
               /* if (1 EQ 2) trifft nie zu!!
                  Aber: Die beschreibende Information aus den Lookup-
                        Tabellen wird zur Compile-Zeit gelesen
               */
               set Daten.Customer_dim         (keep=Customer_ID
                                                    Customer_Name
                                                    Customer_Age_Group
                                                    Customer_Type);
               set Daten.Employee_Addresses   (keep=Employee_ID
                                                    Employee_Name
                                                    Country City
                                    rename=(City     = Employee_City
                                            Country  = Employee_Country
                                           )
                                           );
               set Daten.Product_dim          (keep=Product_ID
                                                    Product_Line
                                                    Product_Category
                                                    Product_Group
                                                    Product_Name);
            end;

            /* Hash-Objekt für Zugriff auf Customer_dim deklarieren
            */
            declare hash hCustomers (dataset: "Daten.Customer_dim");
            hCustomers.definekey ("Customer_ID");
            hCustomers.definedata("Customer_Name"
                                , "Customer_Age_Group"
                                , "Customer_Type");
            hCustomers.definedone();

            /* Hash-Objekt für Zugriff auf Employee_Addresses
               deklarieren */
```

```
   declare hash hEmployees (dataset:
               "Daten.Employee_Addresses
                    (Rename=(City     = Employee_City
                             Country  = Employee_Country)"
                           );
   hEmployees.definekey ("Employee_ID");
   hEmployees.definedata("Employee_Name"
                       , "Employee_Country"
                       , "Employee_City");
   hEmployees.definedone();

   /* Hash-Objekt für Zugriff auf Product_dim deklarieren */
   declare hash hProducts (dataset: "Daten.Product_dim");
   hProducts.definekey ("Product_ID");
   hProducts.definedata("Product_Line"
                      , "Product_Category"
                      , "Product_Group"
                      , "Product_Name");
   hProducts.definedone();
end; /* IF _N_ EQ 1 */

/* Gibt es die aktuelle Customer_ID in der
   Nachschlagetabelle? */
iRC1 = hCustomers.find();
IF (iRC1 NE 0) THEN DO;
   Output Unknown_Customer;
END;

/* Gibt es die aktuelle Employee_ID in der
   Nachschlagetabelle? */
iRC2 = hEmployees.find();
IF (iRC2 NE 0) THEN DO;
   Output Unknown_Employee;
END;

/* Gibt es die aktuelle Employee_ID in der
   Nachschlagetabelle? */
iRC3 = hProducts.find();
IF (iRC3 NE 0) THEN DO;
   Output Unknown_Product;
END;

IF ((iRC1 EQ 0) AND (iRC2 EQ 0) AND (iRC3 EQ 0)) THEN DO;
   /* -  Alle weiteren Berechungen hier eintragen
      ...
      ...
   */
   Output Order_Fact_Complete;
END;

_ERROR_ = 0; /* Fehler im DATA STEP zurücksetzen */
DROP iRC1-iRC3;
```

```
run;

TITLE 'Komplette Informationen zu Bestellungen';
proc print
   data = Order_Fact_Complete (Obs  = 20);
   Var   Order_ID Customer_ID Customer_Name
         Customer_Age_Group Customer_Type
         Employee_ID Employee_Name Employee_Country
         Employee_City Product_ID Product_Line
         Product_Category Product_Group Product_Name
         Order_Date Total_Retail_Price
   ;
run;
TITLE;
```

3 Schlussbemerkungen

- Überlegungen zu Programm-Performance sind immer dann zu treffen, wenn …
  - … Programmläufe sich wiederholen
  - … andere Benutzer auf die Programmausführung warten

- Grundsätzlich kann man folgendes immer tun
  - SAS Optionen zur Performancesteigerung immer aktivieren und nur im Einzelfall abschalten
    - COMPRESS=YES
    - SQLNOREMERGE
    - NOSORTEQUALS
  - Immer versuchen mehrere Aktionen in einem Schritt zu erledigen
  - I/O entlasten durch …
    - … möglichst wenige Zwischendateien speichern
    - … nur benötigte Variablen und Datensätze behalten

- Wenn es ganz kritisch wird
  - Man kann nicht generell sagen "PROC SQL ist besser als DATA STEP" und umgekehrt
  - Im Einzelfall unterschiedliche Varianten testen
  - Einsatz der Scalable Performance Data Engine (SPDE) überdenken
    - Intelligentes Speicherformat für große Datasets