Data Step oder PROC SQL – Was soll ich nehmen?

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

Zusammenfassung

Jeder SAS Programmierer wird die beiden maßgeblichen Techniken kennen, um mit SAS auf Daten zuzugreifen und diese zu manipulieren. Das sind der Data Step und der PROC SQL. Sicher wird es auch so sein, dass der eine oder die andere mit der Zeit eine gewisse Vorliebe für eine der beiden Methoden entwickelt hat.
Neben einer Darstellung der Grundprinzipien von Data Step und PROC SQL sollen in diesem Tutorium anhand von ausgewählten Problemlösungen die jeweiligen Vor- und Nachteile herausgestellt werden. Fragen wie „Wann nimmt man was?“, „Was geht nicht?“ oder „Was geht besonders gut?“ werden dabei auch beantwortet.
Das Tutorium soll die Anwesenden dazu ermutigen, auch mal den anderen Weg zu gehen. Denn letztlich ist es auch in der Programmierung mit SAS so, dass eine gesunde Mischung zu den besten Ergebnissen führt.
Die Beispiele basieren auf SAS in der Version 9.2.

Schlüsselwörter: DATA Step, PROC SQL, HashTable, Performance

1 Inhalt des Tutoriums

Es wird davon ausgegangen, dass beide Verfahren grundsätzlich bereits bekannt sind.
Im Rahmen dieses Beitrags kann auch keinesfalls eine umfassende Beschreibung aller Möglichkeiten gegeben werden, die der DATA Step oder der PROC SQL den SAS-Entwicklern bieten.
Im ersten Abschnitt wird deshalb ganz kurz umrissen, wozu die beiden Verfahren dienen und wie die entsprechenden Programmkonstrukte aussehen.
Danach wird ein Beispielprojekt skizziert, in dessen Rahmen unterschiedliche, konkrete Aufgaben auszuführen sind.
Jeder Arbeitsschritt wird zunächst kurz formal umrissen. Es schließen sich eine Empfehlung für ein bestimmtes Verfahren und vor allem ein konkretes Programmierbeispiel an.

2 DATA Step und PROC SQL – Wie und Wozu?

2.1 Grundsätzliche Aufgaben für DATA Step und PROC SQL

Beide Verfahren werden in SAS-Projekten insbesondere im Zuge der Datenaufbereitung angewendet.
Die grundsätzlichen Aufgaben sind:

- Vorhandene Daten einlesen
  - Rohdaten (nur mit DATA Step)
    - Text-Dateien, CSV-Dateien, Binärdateien, …
  - SAS Datasets
  - Datenbank-Tabellen
    - Oracle, MS EXCEL , OLEDB, …
- Neue SAS Datasets (oder Datenbank-Tabellen) anlegen
- Bestehende SAS Datasets (oder Datenbank-Tabellen) befüllen
- Datenmanipulationen durchführen
  - Daten selektieren
  - (Neue) Datenwerte berechnen
  - Daten gruppieren
  - Daten aggregieren
  - Daten nachschlagen

2.2 Was ist ein DATA Step?

Der DATA Step lässt sich mit folgenden Schlagworten charakterisieren:

- Der DATA Step ist das Herzstück von SAS Base.
- Er besteht aus einer Gruppe von zulässigen SAS Anweisungen, die zwischen einem DATA-Statement und einem RUN-Statement eingefügt werden dürfen.
- Er ist die SAS-Standardmethode für Datenmanipulation

Es folgt eine extrem vereinfachte Darstellung eines DATA Steps, mit dem ein neues Dataset „Target“ dadurch erzeugt wird, dass die Datensätze eines bestehenden Datasets „Source“ der Reihe nach eingelesen und verarbeitet werden.

```
DATA Target;
   SET Source;
   /*
   -  Satzweise Verarbeitung der
      eingelesenen Daten in
      einer Schleife
   -  Dabei
      -  Neue Variablen anlegen
      -  Prozessflusssteuerung
      -  Bedingte Ausführung von
         Statements
   */
   OUTPUT;
RUN;
```

2.3 Was ist ein PROC SQL?

SQL, also Structured Query Language, ist die standardisierte Abfragesprache für Datenbanken. PROC SQL implementiert diesen Standard für SAS, wobei es SAS-spezifische Erweiterungen (Formate, …) gibt.
Ein typischer PROC SQL hat das folgende Aussehen:

```
PROC SQL;
   CREATE TABLE Target AS
      SELECT    {Var-Liste}
      FROM      Source
      <WHERE    {Bedingungen}>
      <GROUP BY {Gruppier-Vars}>
      <HAVING   {Bedingungen}>
      <ORDER BY {Sortier-Vars}
   ;
QUIT;
```

Der dargestellte Code ist dazu geeignet, ein neues Dataset „Target“ zu erzeugen, wobei aus dem bestehenden Dataset „Source“ die Variablen {Var-Liste} selektiert werden sollen und nur diejenigen Datensätze übernommen werden, die den im WHERE-Statement angegebenen {Bedingungen} entsprechen.
Mittels GROUP BY und HAVING können Daten gruppiert abgefragt werden, so dass man z.B. nur einen Datensatz pro Abteilung erhält, der die Summe aller Gehälter, die alle Beschäftigten der Abteilung zusammen erhalten, ausweist. Diesen Vorgang nennt man „Datenaggregation“.
In jedem Fall lassen sich die Ergebnisdatensätze mit Hilfe des ORDER BY-Statements in der gewünschten Form sortieren.

3 Das Tutoriums-Projekt

Im Rahmen des Tutoriums-Projekts sollen Daten aus dem Bereich der Fußball Bundesliga aufbereitet und ausgewertet werden.
Ausgangspunkt sind 2 Textdateien mit den Spielergebnissen der kompletten Saison 2009/2010 und der Hinrunde 2010/2011.

3.1 Ausgangslage

Bei den vorliegenden Textdateien handelt es sich um semikolon-separierte Dateien, bei denen in der ersten Zeile Spaltenüberschriften aufgeführt sind.
Es gibt je Paarung eine Datenzeile u. a. mit Angaben zum Spieldatum, dem Heimverein, dem Gastverein, der Anzahl der von beiden Vereinen geschossenen Tore, sowie dem Halbzeitstand und dem Totoergebnis.
Am Zeilenanfang finden sich Leerzeichen und zwischen zwei Datenzeilen befindet sich immer eine Leerzeile.

```
Land;Liga;Saison;AnzahlSpieltage;AnzahlMannschaften;Spieltag;SpielAmSpieltag;Datum;Uhr
Deutschland;1. Bundesliga;2009/10;34;18;1;1;07.08.2009;20:40;VfL Wolfsburg;VfB St
Deutschland;1. Bundesliga;2009/10;34;18;1;2;08.08.2009;15:30;Borussia Dortmund;1
Deutschland;1. Bundesliga;2009/10;34;18;1;3;08.08.2009;15:30;1.FC Nürnberg;FC Sch
Deutschland;1. Bundesliga;2009/10;34;18;1;4;08.08.2009;15:30;Werder Bremen;Eintra
Deutschland;1. Bundesliga;2009/10;34;18;1;5;08.08.2009;15:30;Hertha BSC;Hannover
```

Abbildung 1: Layout der Rohdatendatei

3.2 Projekt-Ziele

Andreas Brehme
Uns steht ein hartes Programm ins Gesicht

Es sollen Auswertungen möglich sein, die den Verlauf einer Saison bzw. den Saisonverlauf für einen auszuwählenden Verein beschreiben:

- Grafische Darstellung des Saisonverlauf meines Lieblingsvereins
- Bundesligatabelle je Spieltag über eine ganze Saison
- Zusätzliche, spezielle Statistiken
  - Welches Team schoss die meisten Tore in einem Spiel?
  - In welchem Spiel fielen die meisten Tore?
  - …

3.3 Beispielauswertungen

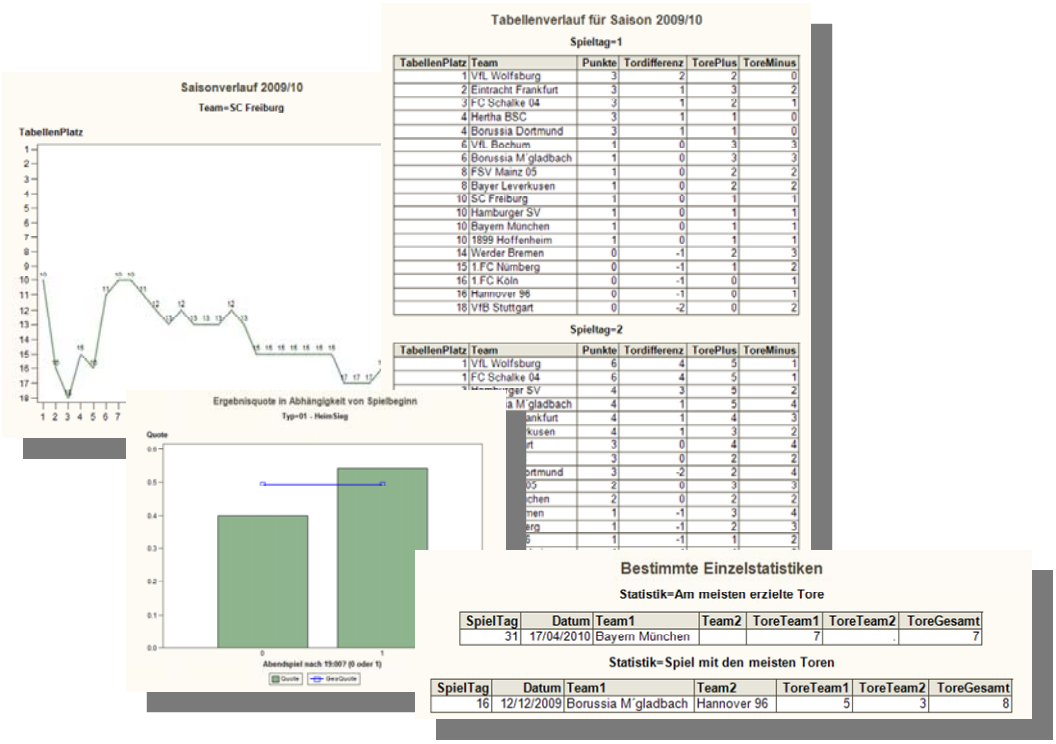

Abbildung 2: Beispiele für geplante Auswertungen

3.4 Die Arbeitsschritte des Tutoriums-Projekts

Die folgenden Arbeitsschritte müssen ausgeführt werden, damit schlussendlich die gewünschten Auswertungen zur Verfügung stehen.

- Vorbereitung
  - Lookup-Tabelle mit allen Mannschaften je Saison anlegen
- Einlesen der Saisonergebnisse 2009/2010
- Hinzufügen von zusätzlichen Spalten
  - PunkteHeim, PunkteGast, …
- Hinzufügen der Saisonergebnisse 2010/2011
- Neue Tabelle „LigaTabelle“ anlegen
  - Eine Zeile je Spieltag und Mannschaft mit dem jeweiligen Punkte- und Torestand
- Auswertungen durchführen
  - Höchste Tordifferenz, Toranzahl, …
- Berichte erzeugen

4 Die Arbeitsschritte im Einzelnen

In diesem Kapitel werden die einzelnen Arbeitsschritte beschrieben. Alle Beispielprogramme und Rohdateien sind beim Autor erhältlich.
Als Vorbereitung muss der folgende Initialisierungs-Code submittiert werden. Der Pfad zum Wurzelverzeichnis muss entsprechend angepasst werden. Unterhalb dieses Verzeichnisses muss ein Verzeichnis namens „DATEN“ existieren, in dem sich die einzulesenden Rohdaten befinden müssen und in dem dann über den Libref „LIGA“ die Zieldatasets gespeichert werden.

- Initialisierungsprogramm

```
%LET g_sProjectRoot  = C:\HMS\Vortraege\KSFE 2011\02_SAS;
%LET g_sDataPath     = &g_sProjectRoot.\Daten;
%LET g_sProgramPath  = &g_sProjectRoot.;
Libname Liga "&g_sDataPath.";
```

4.1 Metatabellen mit statischen Daten anlegen

- Szenario
  - Metatabellen werden in Projekten immer benötigt
    - Häufig als Lookup-Tabellen, die Beschreibungen zu in Datasets verwendeten Codes enthalten
    - Verwendung auch zur Validierung von bestimmten Werten
  - Oft sind die Daten statisch bzw. ändern sich nur sehr selten
    - Z.B. Namen der einzelnen Teams der einzelnen Bundesligasaisons
- Zur Verfügung stehende Techniken
  - DATA Step mit Zuweisen der Werte zu den Variablen und OUTPUT-Statement

  - DATA Step mit INFILE DATALINES4
  - PROC SQL mit CREATE TABLE und anschließendem INSERT-Statement
- Empfehlung für/ Vorteile der anzuwendende(n) Technik
  - Keine der Varianten hat herausragende Vor- oder Nachteile
- TIPP
  - Metatabellen immer mit Hilfe eines Programm-Scripts erstellen, in dem das komplette Dataset immer neu geschrieben und gefüllt wird.
  - Dadurch wird nicht zuletzt die Historisierung von Ständen erleichtert
- Spielstand nach diesem Arbeitsschritt
  - 0 : 0
- Beispielprogramm (gekürzt!)

```
/**
   Lookuptabelle für Teams je Saison anlegen, wobei die Daten
   statisch eingegeben werden
   -  Die einzelnen Varianten haben keine echten Vor- oder
      Nachteile
   -  !!! Extrem wichtig ist es aber, die Struktur des Datasets
      zu Beginn explizit zu definieren
/**/
/**   Alternative 1: DATA Step mit Output
      -  Vorteile
         -  Anlegen des Datasets mit
            SAS/BASE Standard-Statements
            -  LENGTH
            -  FORMAT
            -  ...
            -  Oder gleich ATTRIB
         -  Einfach erweiterbar, wenn mal mehr gemacht werden
            muss
      -  Nachteile
         -  Länglicher Code
            -  Jede Wert-Zuweisung ist ein eigenes Statement
            -  Jeder Datensatz muss explizit mit OUTPUT in das
               Ziel geschrieben werden
/**/
DATA Liga.SaisonTeams;
   LENGTH
      Saison           $ 7
      Team             $ 30
   ;
   Saison   = '2009/10';
   Team     = '1.FC Köln';
   Output;
   Saison   = '2009/10';
   Team     = '1.FC Nürnberg';
   Output;
   Saison   = '2009/10';
   Team     = '1899 Hoffenheim';
```

```
   Output;

/* ... */

   Output;
   Saison    = '2009/10';
   Team      = 'Werder Bremen';
   Output;
   Saison    = '2010/11';
   Team      = '1.FC Kaiserslautern';
   Output;
   Saison    = '2010/11';
   Team      = '1.FC Köln';
   Output;

/* ... */

   Output;
   Saison    = '2010/11';
   Team      = 'VfL Wolfsburg';
   Output;
   Saison    = '2010/11';
   Team      = 'Werder Bremen';
   Output;
RUN;


/*************************************************************/
/*************************************************************/
/**   Alternative 2: DATA Step mit DATALINES-Statement
      -  Vorteile
         -  Anlegen des Datasets mit
            SAS/BASE Standard-Statements
            -  LENGTH
            -  FORMAT
            -  ...
            -  Oder gleich ATTRIB
         -  Es werden einfach nur die Datenwerte eingetragen
      -  Nachteile
         -  Weniger gebräuchliche Syntax
/**/
/**/
DATA Liga.SaisonTeams;
   LENGTH
      Saison            $ 7
      Team              $ 30
   ;
   INFILE DATALINES4
            DLM          = ';'
            MISSOVER
            DSD
   ;
```

```
   INPUT
      Saison
      Team
   ;
DATALINES4;
2009/10;1.FC Köln
2009/10;1.FC Nürnberg
2009/10;1899 Hoffenheim
2009/10;...
2009/10;Werder Bremen
2010/11;1.FC Kaiserslautern
2010/11;1.FC Köln
2010/11;...
2010/11;VfL Wolfsburg
2010/11;Werder Bremen
;;;;
RUN;
/**/


/*************************************************************/
/*************************************************************/
/**   Alternative 3: PROC SQL mit CREATE und
      anschließendem INSERT
      -  Vorteile
         -  Keine, außer man ist SQL-Spezialist
      -  Nachteile
         -  Zwei separate Schritte (Anlegen und Füllen) sind
            notwendig
/**/
/**/
PROC SQL;
   CREATE TABLE Liga.SaisonTeams
      (     Saison            CHAR (7)
         ,  Team              CHAR (30)
      )
   ;
   INSERT INTO Liga.SaisonTeams
      (Saison, Team)
      VALUES ('2009/10', '1.FC Köln')
      VALUES ('2009/10', '1.FC Nürnberg')
      VALUES ('2009/10', '1899 Hoffenheim')

/* ... */

      VALUES ('2009/10', 'Werder Bremen')
      VALUES ('2010/11', '1.FC Kaiserslautern')
      VALUES ('2010/11', '1.FC Köln')

/* ... */

      VALUES ('2010/11', 'VfL Wolfsburg')
```

```
        VALUES ('2010/11', 'Werder Bremen')
    ;
QUIT;
/**/
```

4.2 Rohdaten aus externer Datei einlesen

- Szenario
  - Die Spielergebnisse müssen aus Textdateien eingelesen werden
  - Die einzelnen Werte sind durch Semikolons voneinander getrennt
- Zur Verfügung stehende Techniken
  - DATA Step mit INFILE-Statement
- Empfehlung für/ Vorteile der anzuwendende(n) Technik
  - DATA Step, da einzige Alternative
  - Vorteile DATA Step
    - Die ganze Mächtigkeit des DATA Steps steht zur Verfügung
    - Schon beim Einlesen hat man die Möglichkeit, Umrechnungen usw. durchzuführen
- TIPP
  - Wenn man Einfluss nehmen kann, dann immer eine kommaseparierte Datei anfordern statt einer Datei mit Werten in fester Spaltenbreite
  - Alle Variablen in der gewünschten Zielreihenfolge zu Beginn deklarieren, ggf. mit FORMAT- und INFORMAT-Statement.
    - Danach nur noch einlesen mit INPUT-Statement.
- Spielstand nach diesem Arbeitsschritt
  - 1 : 0 für DATA Step
- Beispielprogramm

```
/**
   Einlesen der Rohdaten mit den Saisonergebnissen 2009/2010
   -  Die erste Zeile enthält Spaltenüberschriften
   -  Danach und nach jeder Datenzeile folgt immer eine
      Leerzeile
/**/
/**   Alternative 1: DATA Step mit INFILE
/**/
DATA Liga.Results;
   LENGTH
      Land                $ 15
      Liga                $ 20
      Saison              $ 7
      AnzahlSpieltage     8
      AnzahlMannschaften  8
      Spieltag            8
      SpielAmSpieltag     8
      Datum               8
      Uhrzeit             8
      HeimVerein          $ 19
```

```
      GastVerein          $ 19
      ToreHeim            8
      ToreGast            8
      HalbzeitHeim        8
      HalbzeitGast        8
      TotoResult          $ 1
   ;
   FORMAT
      Datum            DDMMYY10.
      Uhrzeit          TIME8.
   ;
   INFORMAT
      Datum            DDMMYY10.
      Uhrzeit          TIME8.
   ;
   INFILE "&g_sDataPath.\Tabelle_2009__Komplett.csv"
            DLM=';'
            MISSOVER
            DSD
   ;
   IF (_N_ GT 1) THEN DO;
      INPUT
         Land
         Liga
         Saison
         AnzahlSpieltage
         AnzahlMannschaften
         Spieltag
         SpielAmSpieltag
         Datum
         Uhrzeit
         HeimVerein
         GastVerein
         ToreHeim
         ToreGast
         HalbzeitHeim
         HalbzeitGast
         TotoResult
      ;
      IF (HeimVerein NE "") THEN DO;
         Output;
      END; /* IF Keine Leerzeile */
   END; /* Lesen ab der zweiten Zeile */
   ELSE DO;
      /* Einlesen, aber nicht verarbeiten der ersten
         Zeile mit den Spaltenüberschriften */
      INPUT ;
   END;

   Drop Land Liga AnzahlMannschaften AnzahlSpieltage;
RUN;
```

4.3 Erweitern der Struktur eines bestehenden Datasets

- Szenario
  - Hinzufügen neuer Spalten zu den Spielergebnissen, deren Werte sich aus den bestehenden Spalten berechnen lassen
  - Ziel dabei ist es, spätere Zugriffe zu vereinfachen
  - Beispiel:
    - Wie viele Punkte erreichte die Heimmannschaft durch das jeweilige Spiel?
    - Handelte es sich um ein Abendspiel?
- Zur Verfügung stehende Techniken
  - DATA Step mit SET auf die Ursprungsdatei
  - PROC SQL mit CREATE TABLE für Zwischendatei und SELECT auf die Ursprungsdatei
- Empfehlung für/ Vorteile der anzuwendende(n) Technik
  - DATA Step
    - Code gut strukturierbar und gut lesbar
    - Berechnung der neuen Werte gut steuerbar
    - Die Datei kann in einem Step mit sich selbst überschrieben werden, was beim PROC SQL nicht geht (Erzeugen einer neuen Datei, Löschen des Originals und Umbenennen nötig)
- Spielstand nach diesem Arbeitsschritt
  - DATA Step erhöht auf 2 : 0
- Beispielprogramm

```
/**
   Hinzufügen neuer Spalten je Spielpaarung, sowie Ermittlung
   der Werte für die neuen Spalten
   -  Dadurch soll es später leichter werden, die
      Bundesliga-Tabelle zu berechnen und bestimmte Statistiken
      zu ermitteln
/**/
/**   Alternative 1: DATA Step mit SET auf die Ursprungsdatei
/**/
DATA Liga.Results;
   SET Liga.Results;

   LENGTH
      PunkteHeim              4
      PunkteGast              4
      HalbzeitPunkteHeim      4
      HalbzeitPunkteGast      4
      SiegHeim                4
      RemisHeim               4
      NiederlageHeim          4
      SiegGast                4
      RemisGast               4
      NiederlageGast          4
```

```
      AbendSpiel              4
      WochenTag               4
   ;
   LABEL
      PunkteHeim              = "Punkte für Heimmannschaft"
      PunkteGast              = "Punkte für Gastmannschaft"
      HalbzeitPunkteHeim      = "Punkte zur Halbzeit für Heim-
mannschaft"
      HalbzeitPunkteGast      = "Punkte zur Halbzeit für Gast-
mannschaft"
      SiegHeim                = "Sieg für Heimmannschaft (0
oder 1)"
      RemisHeim               = "Unentschieden für Heimmann-
schaft (0 oder 1)"
      NiederlageHeim          = "Niederlage für Heimmannschaft
(0 oder 1)"
      SiegGast                = "Sieg für Gastmannschaft (0
oder 1)"
      RemisGast               = "Unentschieden für Gastmann-
schaft (0 oder 1)"
      NiederlageGast          = "Niederlage für Gastmannschaft
(0 oder 1)"
      AbendSpiel              = "Abendspiel nach 19:00? (0 oder
1)"
      WochenTag               = "Wochentag (6=Samstag)"
   ;

   /* Wochentag und Abendspiel ermitteln */
   AbendSpiel           = (Uhrzeit GT '19:00't);
   Wochentag            = IfN (WeekDay (Datum) EQ 1
                              , 7
                              , WeekDay (Datum) - 1);

   /* Erst einmal alles auf 0 setzen */
   PunkteHeim           = 0;
   PunkteGast           = 0;
   HalbzeitPunkteHeim   = 0;
   HalbzeitPunkteGast   = 0;
   SiegHeim             = 0;
   RemisHeim            = 0;
   NiederlageHeim       = 0;
   SiegGast             = 0;
   RemisGast            = 0;
   NiederlageGast       = 0;

   /* Daten für Halbzeit-Tabelle */
   IF (HalbzeitHeim GT HalbzeitGast) THEN DO;
      HalbzeitPunkteHeim      = 3;
   END; /* IF zur Halbzeit: Heimsieg */
   ELSE DO;
      IF (HalbzeitHeim EQ HalbzeitGast) THEN DO;
         HalbzeitPunkteGast   = 1;
```

```
      END; /* IF zur Halbzeit: Unentschieden */
      ELSE DO;
         HalbzeitPunkteGast   = 3;
      END; /* ELSE: Wenn zur Halbzeit: Gästesieg */
   END; /* IF zur Halbzeit: KEIN Unentschieden */

   /* Endresultat: Sieg und Niederlage auswerten */
   IF (ToreHeim GT ToreGast) THEN DO;
      PunkteHeim           = 3;
      SiegHeim             = 1;
      NiederlageGast       = 1;
   END; /* IF Heimsieg */
   ELSE DO;
      IF (ToreHeim EQ ToreGast) THEN DO;
         PunkteHeim           = 1;
         PunkteGast           = 1;
         RemisHeim            = 1;
         RemisGast            = 1;
      END; /* IF Unentschieden */
      ELSE DO;
         PunkteGast           = 3;
         NiederlageHeim       = 1;
         SiegGast             = 1;
      END; /* ELSE: Wenn Gästesieg */
   END; /* IF KEIN Unentschieden */

RUN;


/**   Alternative 2: PROC SQL mit vielen CASE-Anweisungen
/**/
/**
* Zu Beginn die Ausgangssituation nach dem
  Einlesen wieder herstellen;
%INCLUDE "&g_sProgramPath.\11-Saison 2009_2010 einlesen.sas";
PROC SQL NOPRINT;
   CREATE TABLE Liga.Results_02 AS
      SELECT   a.*
             ,  CASE
                   WHEN (ToreHeim GT ToreGast) THEN 3
                   WHEN (ToreHeim EQ ToreGast) THEN 1
                   ELSE 0
                END                        AS PunkteHeim
LENGTH=4     LABEL="Punkte für Heimmannschaft"
             ,  CASE
                   WHEN (ToreHeim GT ToreGast) THEN 0
                   WHEN (ToreHeim EQ ToreGast) THEN 1
                   ELSE 3
                END                        AS PunkteGast
LENGTH=4     LABEL="Punkte für Gastmannschaft"
             ,  CASE
                   WHEN (HalbzeitHeim GT HalbzeitGast) THEN 3
```

```
                  WHEN (HalbzeitHeim EQ HalbzeitGast) THEN 1
                  ELSE 0
               END                      AS HalbzeitPunkteHeim
LENGTH=4       LABEL="Punkte zur Halbzeit für Heimmannschaft"
               ,  CASE
                  WHEN (HalbzeitHeim GT HalbzeitGast) THEN 0
                  WHEN (HalbzeitHeim EQ HalbzeitGast) THEN 1
                  ELSE 3
               END                      AS HalbzeitPunkteGast
LENGTH=4       LABEL="Punkte zur Halbzeit für Gastmannschaft"
               ,  (ToreHeim GT ToreGast)  AS SiegHeim
LENGTH=4       LABEL="Sieg für Heimmannschaft (0 oder 1)"
               ,  (ToreHeim EQ ToreGast)  AS RemisHeim
LENGTH=4       LABEL="Unentschieden für Heimmannschaft (0 oder 1)"
               ,  (ToreHeim LT ToreGast)  AS NiederlageHeim
LENGTH=4       LABEL="Niederlage für Heimmannschaft (0 oder 1)"
               ,  (ToreHeim LT ToreGast)  AS SiegGast
LENGTH=4       LABEL="Sieg für Gastmannschaft (0 oder 1)"
               ,  (ToreHeim EQ ToreGast)  AS RemisGast
LENGTH=4       LABEL="Unentschieden für Gastmannschaft (0 oder 1)"
               ,  (ToreHeim GT ToreGast)  AS NiederlageGast
LENGTH=4       LABEL="Niederlage für Gastmannschaft (0 oder 1)"
               ,  (Uhrzeit GT '19:00't)   AS AbendSpiel
LENGTH=4       LABEL="Abendspiel nach 19:00? (0 oder 1)"
               ,  CASE
                  WHEN (WeekDay (Datum) EQ 1)         THEN 7
                  ELSE (WeekDay (Datum) - 1)
               END                      AS WochenTag
LENGTH=4       LABEL="Wochentag (6=Samstag)"
      FROM     Liga.Results      a
   ;
QUIT;

* Jetzt noch die bisherige Zieldatei löschen und die neue,
   erweiterte Struktur umbenennen;
PROC DATASETS NOLIST LIB   = Liga;
   DELETE
      Results
   ;
   CHANGE
      Results_02 = Results
   ;
QUIT;
/**/
```

4.4 Rohdaten aus externer Datei einlesen; Zielstruktur bereits vorhanden

- Szenario
  - Ergebnisse der Saison 2010/2011 sollen gleich in die erweiterte Struktur eingelesen werden.

  - Ziel dieses Einlese-Schritts ist eine Zwischendatei, die anschließend an die bisher bereits eingelesenen Ergebnisse angespielt werden soll.
  - Der Unterschied zu Arbeitsschritt 4.2 ist der, dass es bereits ein Dataset gibt, in dem die Zielstruktur gespeichert ist (siehe TIPP).
- Zur Verfügung stehende Techniken
  - DATA Step mit INFILE-Statement
- Empfehlung für/ Vorteile der anzuwendende(n) Technik
  - DATA Step , da einzige Alternative
- TIPP
  - SET auf das bereits vorhandene Dataset mit der Zielstruktur verwenden
    - Compiler liest die aktuelle Struktur ein und kennt somit die Deklaration der Zielvariablen
  - Dabei im DATA Step durch geeignete Programmierung sicherstellen, dass dieses SET niemals ausgeführt wird
  - Die Möglichkeit der Verwendung von mehreren SET- bzw. INFILE-Statements in einem DATA Step kann dadurch sehr effizient eingesetzt werden, die Definition der Variablen muss nicht noch einmal codiert werden, was sehr fehleranfällig wäre
- Spielstand nach diesem Arbeitsschritt
  - 3 : 0 für DATA Step
- Beispielprogramm

```
/**
   Einlesen der Rohdaten mit den Saisonergebnissen 2010/2011
   -  Hinweis: Aktuell nur bis incl. Spieltag 17
   -  Die erste Zeile enthält Spaltenüberschriften
   -  Danach und nach jeder Datenzeile folgt immer eine
      Leerzeile
   -  Spiele, bei denen das Ergebnis noch nicht bekannt ist,
      haben in TotoResult den Wert 'X'
   -  Die zusätzlichen Datenwerte, die der Zielstruktur im
      letzten Schritt hinzugefügt worden sind, werden direkt
      beim Einlesen ermittelt
   -  Das Ergebnis steht zunächst in einer WORK-Datei und soll
      im nächsten Schritt an die bisherigen Ergebnisse
      angespielt werden
/**/
/**   Alternative 1: DATA Step mit INFILE
/**/
DATA WORK.Results_2010_2011;

   IF (1 EQ 2) THEN DO;
      /* -  Der Compiler findet als erstes das SET-Statement
            und liest somit die Struktur der bisherigen
            Ergebnis-Datei ein und bereitet sie im
            Program Data Verctor (PDV) auf
         -  Da aber zur Laufzeit 1 niemals gleich 2 sein wird,
            wird das SET Statement niemals ausgeführt :-)
```

```
   */
   SET Liga.Results (OBS = 0);
END;

/* Die Variablen, die nicht mehr in der Zieldatei enthalten
   sind, müssen hier deklariert werden */
LENGTH
   Land                 $ 15
   Liga                 $ 20
   AnzahlSpieltage      8
   AnzahlMannschaften   8
;

INFILE "&g_sDataPath.\Tabelle_2010__Komplett.csv"
         DLM=';'
         MISSOVER
         DSD
;
IF (_N_ GT 1) THEN DO;
   INPUT
      Land
      Liga
      Saison
      AnzahlSpieltage
      AnzahlMannschaften
      Spieltag
      SpielAmSpieltag
      Datum
      Uhrzeit
      HeimVerein
      GastVerein
      ToreHeim
      ToreGast
      HalbzeitHeim
      HalbzeitGast
      TotoResult
   ;

   /* Leerzeilen und noch nicht durchgeführte Spiele werden
      übersprungen */
   IF (HeimVerein NE "") AND (TotoResult NE 'X') THEN DO;

      /* Wochentag und Abendspiel ermitteln */
      AbendSpiel           = (Uhrzeit GT '19:00't);
      Wochentag            = WeekDay (Datum);

      /* Erst einmal alles auf 0 setzen */
      PunkteHeim           = 0;
      PunkteGast           = 0;
      HalbzeitPunkteHeim   = 0;
      HalbzeitPunkteGast   = 0;
      SiegHeim             = 0;
```

```
            RemisHeim            = 0;
            NiederlageHeim       = 0;
            SiegGast             = 0;
            RemisGast            = 0;
            NiederlageGast       = 0;

            /* Daten für Halbzeit-Tabelle */
            IF (HalbzeitHeim GT HalbzeitGast) THEN DO;
               HalbzeitPunkteHeim      = 3;
            END; /* IF zur Halbzeit: Heimsieg */
            ELSE DO;
               IF (HalbzeitHeim EQ HalbzeitGast) THEN DO;
                  HalbzeitPunkteGast   = 1;
               END; /* IF zur Halbzeit: Unentschieden */
               ELSE DO;
                  HalbzeitPunkteGast   = 3;
               END; /* ELSE: Wenn zur Halbzeit: Gästesieg */
            END; /* IF zur Halbzeit: KEIN Unentschieden */

            /* Endresultat: Sieg und Niederlage auswerten */
            IF (ToreHeim GT ToreGast) THEN DO;
               PunkteHeim           = 3;
               SiegHeim             = 1;
               NiederlageGast       = 1;
            END; /* IF Heimsieg */
            ELSE DO;
               IF (ToreHeim EQ ToreGast) THEN DO;
                  PunkteHeim           = 1;
                  PunkteGast           = 1;
                  RemisHeim            = 1;
                  RemisGast            = 1;
               END; /* IF Unentschieden */
               ELSE DO;
                  PunkteGast           = 3;
                  NiederlageHeim       = 1;
                  SiegGast             = 1;
               END; /* ELSE: Wenn Gästesieg */
            END; /* IF KEIN Unentschieden */

            Output;
         END; /* IF Keine Leerzeile */
      END; /* Lesen ab der zweiten Zeile */
      ELSE DO;
         /* Einlesen, aber nicht verarbeiten der ersten Zeile
            mit den Spaltenüberschriften */
         INPUT ;
      END;

      Drop Land Liga AnzahlMannschaften AnzahlSpieltage;
RUN;
```

4.5 Anhängen der Zeilen eines Datasets an ein Anderes

- Szenario
  - Die soeben eingelesenen Ergebnisse der Hinrunde 2010/2011 sollen an das bestehende Dataset LIGA.RESULTS angehängt werden
    - Es wurde im Vorfeld sichergestellt, dass die Strukturen beider Datasets identisch sind
    - In diesem Schritt müssen also keinerlei zusätzliche Manipulationen durchgeführt werden
- Zur Verfügung stehende Techniken
  - PROC SQL mit INSERT-Statement, sowie SELECT-Statement, um die anzuhängenden Datensätze zu selektieren
  - DATA Step mit einem SET-Statement für beide Datasets
  - PROC APPEND als spezialisierte Prozedur
    - Geht nicht in die Bewertung ein
- Empfehlung für/ Vorteile der anzuwendende(n) Technik
  - PROC SQL
  - Nachteile DATA Step
    - Die Daten beider Datasets müssen komplett gelesen und geschrieben werden
    - Indizes und Constraints gehen durch das Neuschreiben verloren und müssen explizit neu angelegt werden
- Spielstand nach diesem Arbeitsschritt
  - PROC SQL kann verkürzen; nur noch 3 : 1 für DATA Step
- PROC APPEND
  - Vorteile: Es müssen nur die Daten der anzuhängenden Datei gelesen und geschrieben werden, Indices und Constraints in der Zieldatei bleiben erhalten
  - Nachteile: Wenn die Strukturen der Quell- und Zieldatei nicht identisch sind, muss man die OPTION FORCE verwenden und riskiert dabei Datenverlust
- Beispielprogramm

```
/**
   Einlesen der Rohdaten mit den Saisonergebnissen 2010/2011
   -  Hinweis: Aktuell nur bis incl. Spieltag 17
   -  Die erste Zeile enthält Spaltenüberschriften
   -  Danach und nach jeder Datenzeile folgt immer eine
      Leerzeile
   -  Die zusätzlichen Datenwerte werden direkt beim Einlesen
      ermittelt
   -  Das Ergebnis steht zunächst in einer WORK-Datei und soll
      im nächsten Schritt an die bisherigen Ergebnisse
      angespielt werden
/**/
/**   Alternative 1: PROC SQL mit INSERT
/**/
```

```
PROC SQL NOPRINT;
   INSERT INTO Liga.Results
      SELECT * FROM WORK.Results_2010_2011
   ;
QUIT;

/**   Alternative 2: DATA Step mit einem SET für beide Dateien
/**
DATA LIGA.Results;
   SET
      Liga.Results
      WORK.Results_2010_2011
   ;
RUN;
/**/

/**   Alternative 3: PROC APPEND
/**
PROC APPEND
   BASE   = Liga.Results
   DATA   = WORK.Results_2010_2011
   ;
QUIT;
/**/
```

4.6 Metatabellen mit Daten aus anderen Datasets anlegen

- Szenario
  - Für den nächsten Schritt wird die Liste aller vorkommenden Spieltag-Nummern einer beliebigen Saison benötigt
    - Ein Dataset mit einer Variablen, die die Spieltag-Nummer enthält
    - Das Dataset soll aufsteigend nach Spieltag-Nummer sortiert sein
- Zur Verfügung stehende Techniken
  - PROC SQL mit CEATE TABLE und SELECT DISTINCT-Statement
  - DATA Step, im Beispiel mit statischer Erzeugung der Spieltags-Nummern
  - PROC SORT mit NODUPKEY als spezialisierte Prozedur
    - Geht nicht in die Bewertung ein
- Empfehlung für/ Vorteile der anzuwendende(n) Technik
  - PROC SQL
    - Vorteil: Ergebnis kann gleich korrekt sortiert werden
  - Nachteile DATA Step
    - Die Anzahl Spieltage muss bekannt sein. Die vorliegende konkrete Anforderung kann mit einem DATA Step ohne aufwändige Vorbereitung sonst nicht umgesetzt werden
    - Hinweis: Im verwendeten DATA Step-Beispiel wird einfach davon ausgegangen, dass es 34 Spieltage gibt
  - Vorteil PROC SORT: kurz und bündig

- Spielstand nach diesem Arbeitsschritt
  - PROC SQL kommt näher heran; neuer Spielstand: 3 : 2 für DATA Step
- Beispielprogramm

```
/**
   Erzeugen eines neuen Datasets mit allen Spieltagen, die es
   geben kann
   -  Hinweis: Das ist eine Vorbereitung für den nächsten
      Schritt
/**/
/**   Alternative 1: PROC SQL mit SELECT DISTINCT
/**/
PROC SQL NOPRINT;
   CREATE TABLE SpielTage AS
      SELECT   Distinct (SpielTag)
      FROM     Liga.Results
      WHERE    Not Missing (Spieltag)
      ORDER BY SpielTag
   ;
QUIT;

/**   Alternative 2: DATA Step
/**/
Data SpielTage;
   Do SpielTag = 1 To 34;
      Output;
   End;
Run;
/**   Alternative 3: PROC SORT mit NODUPKEY
/**/
PROC SORT NODUPKEY
   DATA  = Liga.Results (KEEP = SpielTag)
   OUT   = SpielTage
   ;
   BY SpielTag;
RUN;
```

4.7 Kartesisches Produkt der Zeilen zweier Datasets

- Szenario
  - Es wird ein Dataset benötigt mit einem Datensatz pro Saison/Mannschaft und Spieltag
    - LIGA.LIGATABELLE
  - Diese Struktur soll gleichzeitig um zusätzliche Informationen erweitert werden wie z.B.
    - Datum, Tabellenplatz, PunkteHeute, PunkteGesamt, …
  - Wir wollen also für jeden Mannschaftseintrag 34 Datensätze erhalten, um den jeweiligen Tabellenplatz usw. dieser Mannschaft am jeweiligen Spieltag speichern zu können

- Zur Verfügung stehende Techniken
  - PROC SQL mit CREATE TABLE und SELECT aus 2 Datasets ohne Angabe von JOIN-Bedingung
  - DATA Step mit zweitem SET und POINT-Option
- Empfehlung für/ Vorteile der anzuwendende(n) Technik
  - PROC SQL
  - Vorteil PROC SQL
    - Das kartesische Produkt ist die grundlegende Arbeitsweise des PROC SQL beim JOIN
    - Dataset kann gleich richtig sortiert werden
  - Nachteile DATA Step
    - Sehr aufwändig zu realisieren
    - Man muss selbst eine Schleife codieren, in der für jeden Datensatz aus Dataset 1 alle Datensätze des Datasets 2 eingelesen werden
    - Man muss selbst dafür Sorge tragen, dass der DATA Step terminiert und keine Endlosschleife resultiert
- TIPP
  - In diesem Ausnahmefall darf bei PROC SQL die SAS NOTE bzgl. des kartesischen Produkts ignoriert werden
- Spielstand nach diesem Arbeitsschritt
  - PROC SQL gleicht aus zum 3 : 3
- Beispielprogramm

```
/**
   Erzeugen eines neuen Datasets mit einem Datensatz pro
   Mannschaft und Spieltag
   -  Hinweis: Man benötigt ein kartesisches Produkt
/**/
/**   Alternative 1: PROC SQL mit SELECT Zugriff auf 2
      Tabellen ohne Join-Bedingung
/**/
PROC SQL NOPRINT;
   CREATE TABLE Liga.LigaTabelle_01 AS
      SELECT   a.Saison
             , b.Spieltag
             , 0          Format=DDMMYY10.    AS Datum
             , a.Team
             , 0                              As TabellenPlatz
             , b.Spieltag                     As Spiele
             , 0                              As PunkteHeute
             , 0                              As TorePlusHeute
             , 0                              As ToreMinusHeute
             , 0                              As Punkte
             , 0                              As Tordifferenz
             , 0                              As TorePlus
             , 0                              As ToreMinus
             , 0                              As OK
      FROM     Liga.SaisonTeams  a
```

```
            ,  SpielTage          b
      ORDER BY a.Saison
            ,  a.Team
            ,  b.Spieltag
   ;
Quit;


/**   Alternative 2: DATA Step mit zweitem SET und POINT-Option
/**
DATA Liga.LigaTabelle_01;
   /* RETAIN, für die gewünschte Variablenreihenfolge */
   RETAIN
      Saison
      Spieltag
      Datum
      Team
   ;

   SET Liga.SaisonTeams (KEEP = Saison Team) END=Letzter1;

   FORMAT
      Datum          DDMMYY10.
   ;
   /* Schleife, in der für jeden Datensatz aus Dataset 1 alle
      Datensätze des Datasets 2 eingelesen werden */
   DO RecNum = 1 TO Anzahl2;
      /* NOBS-Option: Anzahl der zu lesenden Datensätze,
         POINT-Option: Lesen zweites Dataset Satz für Satz */
      SET SpielTage (KEEP=Spieltag) POINT=RecNum NOBS=Anzahl2;

      Datum          = 0;
      TabellenPlatz  = 0;
      Spiele         = Spieltag;
      PunkteHeute    = 0;
      TorePlusHeute  = 0;
      ToreMinusHeute = 0;
      Punkte         = 0;
      Tordifferenz   = 0;
      TorePlus       = 0;
      ToreMinus      = 0;
      OK             = 0;

      OUTPUT;
   END;

   *  -  Wenn der letzte Datensatz aus Dataset 1 verarbeitet
         ist, wird gestoppt.
      -  Sonst: Nächste Iteration im DATA Step
   ;
   IF (Letzter1 EQ 1) THEN DO;
      STOP;
```

```
    END;
    ELSE DO;
       RETURN;
    END;

    DROP RecNum;
RUN;
/**/
```

4.8 Daten eines Datasets aktualisieren mit Werten einer Nachschlagetabelle

- Szenario
  - Im Dataset LIGA.LIGATABELLE sollen pro Saison, Team und Spieltag die erreichten Punkte und Tore kumuliert werden, so dass anschließend die Punktetabelle je Spieltag ermittelt werden kann
    - Benötigt werden aus LIGA.RESULTS: Datum, ToreHeim und ToreGast
  - Problem
    - Einfacher Join geht nicht, da das Team ja Heim- oder Gastmannschaft sein kann
- Zur Verfügung stehende Techniken
  - DATA Step mit BY-Verarbeitung
  - SQL Join bzw. SQL SELECT mit Formaten scheidet aus, da nicht in einem Schritt realisierbar
- Empfehlung für/ Vorteile der anzuwendende(n) Technik
  - DATA Step
  - Vorteile DATA Step
    - Mit HASHTABLE-Lookup können gleichzeitig mehrere Schlüssel abgeglichen und mehrere Zielwerte übernommen werden
    - Die Hashtabelle muss vorher nicht sortiert sein
    - Die Hashtabelle wird in den Hauptspeicher gelesen, wodurch das Nachschlagen sehr schnell geht (Vorsicht bei sehr großen Tabellen wegen Speicherplatz!)
    - Mehrere Hashtabellen sind möglich
    - BY- Verarbeitung im DATA Step ermöglicht Kumulieren
- TIPP
  - Hat man variable Joins, ggf. unterschiedliche Nachschlage-Tabellen mit unterschiedliche Nachschlage-Werten
    - => DATA Step mit HASHTABLE-Lookup
- Spielstand nach diesem Arbeitsschritt
  - Erneute Führung für DATA Step 4 : 3
- Beispielprogramm

```
/**
   Füllen der Ligatabelle mit den Ergebnissen der einzelnen
```

```
      Spieltage, um somit die Daten für die Ermittlung des
      aktuellen Tabellenstand jedes Teams an jedem Spieltag zu
      erhalten
      -  Hinweis: Join geht hier nicht (so richtig), da man das
         Team einmal gegen die Variable 'HeimVerein' und einmal
         gegen die Variable 'GastVerein' mergen muss
      -  Man muss die bestehende Zieldatei überarbeiten, wobei
         die zu verwendenden Werte in WORK.Results
         nachgeschlagen werden müssen
   /**/
   /**   Alternative 1: DATA Step mit Lookup via HASHTABLE
   /**/
   DATA Liga.LigaTabelle_02;
      LENGTH
         bTeamGefunden        3
      ;

      SET Liga.LigaTabelle_01;
      /* Lag-Funktion kann nicht verwendet werden, da der Wert der
         Lag-Variable selbst geändert wird.
         Deshalb: RETAIN, um Punkte und Tore zu kumulieren */
      RETAIN
         PrevPunkte
         PrevTorePlus
         PrevToreMinus
      ;

      BY Saison Team Spieltag;

      IF (_N_ EQ 1) THEN DO;
         IF (1 EQ 2) THEN DO;
            /* -  siehe 4.4 */
            SET Liga.Results (OBS    = 0
                              KEEP   = Saison Spieltag Datum
                                       HeimVerein GastVerein
                                       ToreHeim ToreGast
                             );
         END;

         DECLARE hash luHeim (dataset: "Liga.Results");
         luHeim.definekey  ("Saison", "Spieltag", "HeimVerein");
         luHeim.definedata ("Datum", "ToreHeim", "ToreGast");
         luHeim.definedone ();

         DECLARE hash luGast (dataset: "Liga.Results");
         luGast.definekey  ("Saison", "Spieltag", "GastVerein");
         luGast.definedata ("Datum", "ToreHeim", "ToreGast");
         luGast.definedone ();

      END; /* Hash-Objekt nur bei erster Iteration deklarieren */

      bTeamGefunden     = 0;
```

```
/* -  Zunächst schauen, ob das aktuelle Team am aktuellen
      Spieltag eine Heimmannschaft war.
   -  Ein Treffer wird an iRC=0 erkannt und die
      Nachschlagewerte stehen automatisch in den Variablen
      "Datum", "ToreHeim", "ToreGast" zur Verfügung
*/
HeimVerein  = Team;
iRC = luHeim.find();
IF (iRC EQ 0) THEN DO;
   bTeamGefunden  = 1;
   TorePlusHeute  = ToreHeim;
   ToreMinusHeute = ToreGast;
END; /* Wenn die aktuelle Mannschaft Heimrecht hatte */
ELSE DO;
   /* Nun schauen, ob das aktuelle Team am aktuellen
      Spieltag eine Gastmannschaft war */
   GastVerein  = Team;
   iRC = luGast.find();
   IF (iRC EQ 0) THEN DO;
      bTeamGefunden  = 1;
      TorePlusHeute  = ToreGast;
      ToreMinusHeute = ToreHeim;
   END; /* Wenn die aktuelle Mannschaft Heimrecht hatte */
END; /* ELSE: Wenn die Mannschaft KEIN Heimrecht hatte */

IF (bTeamGefunden GT 0) THEN DO;
   /* Nun werden die neuen Werte aus den nachgeschlagenen
      Werten berechnet */
   IF (TorePlusHeute GT ToreMinusHeute) THEN DO;
      PunkteHeute    = 3;
   END; /* IF Heimsieg */
   ELSE DO;
      IF (TorePlusHeute EQ ToreMinusHeute) THEN DO;
         PunkteHeute    = 1;
      END; /* IF Unentschieden */
      ELSE DO;
         PunkteHeute    = 0;
      END; /* ELSE: Wenn Heimniederlage */
   END; /* ELSE: Wenn KEIN Heimsieg */

   IF (First.Team EQ 1) THEN DO;
      Punkte         = PunkteHeute;
      TorePlus       = TorePlusHeute;
      ToreMinus      = ToreMinusHeute;
   END; /* IF neues Team wird kumuliert */
   ELSE DO;
      Punkte         = Sum (PrevPunkte,    PunkteHeute);
      TorePlus       = Sum (PrevTorePlus,  TorePlusHeute);
      ToreMinus      = Sum (PrevToreMinus, ToreMinusHeute);
   END; /* ELSE: Team-Kumulierung wird fortgesetzt */
```

```
      Tordifferenz   = Sum (TorePlus,        -ToreMinus);

   END; /* Wenn die Nachschlage-Werte für die aktuelle
           Mannschaft am aktuellen Spieltag gefunden wurden */
   ELSE DO;
      /* An aktuellen Spieltag liegt für die aktuelle
         Mannschaft noch kein Spielergebnis in Results vor
      */
      _ERROR_ = 0;
   END; /* ELSE: Wenn für die Mannschaft am Spieltag kein
               Spielergebnis vorliegt */

   /* Registrieren, ob dieser Datensatz
      (Spieltag und Verein) bereits verarbeitet ist */
   OK             = bTeamGefunden;
   PrevPunkte     = Punkte;
   PrevTorePlus   = TorePlus;
   PrevToreMinus  = ToreMinus;

   OUTPUT;

   /* Nicht benötigte Variablen loswerden */
   DROP  iRC bTeamGefunden HeimVerein GastVerein
         ToreHeim ToreGast
         PrevPunkte PrevTorePlus PrevToreMinus
   ;

RUN;

/**
   Abschließend wird die Tabellenplatzierung jedes Teams an
   jedem Spieltag ermittelt.
   Es muss berücksichtigt werden, dass bei Punktgleichheit
   in Verbindung mit gleicher Tordifferenz und in Verbindung
   mit gleicher Anzahl geschossener Tore der gleiche Platz
   eingenommen wird.
   -  Hinweis: Als Vorbereitung wird die Ligatabelle nach
      entsprechenden Variablen sortiert
/**/
/**   Alternative 1: DATA Step mit BY-Verarbeitung
/**/
PROC SORT
   DATA  = Liga.LigaTabelle_02
   OUT   = Liga.LigaTabelle_03;
   BY    Saison
         Spieltag
         Descending     Punkte
         Descending     Tordifferenz
         Descending     TorePlus
   ;
RUN;
```

```
DATA Liga.LigaTabelle;

   SET Liga.LigaTabelle_03;

   BY Saison Spieltag;

   RETAIN
      CurRang           0
      CurTeam           0
   ;

   IF (First.Spieltag EQ 1) THEN DO;
      CurRang     = 0;
      CurTeam     = 0;
   END; /* IF neuer Spieltag beginnt */

   CurTeam        = CurTeam + 1;

   IF ((Punkte NE Lag (Punkte))
         OR (Tordifferenz NE Lag (Tordifferenz))
         OR (TorePlus NE Lag (TorePlus))) THEN DO;
      CurRang        = CurTeam;
   END; /* IF die aktuelle Mannschaft ist NICHT punkt- und
           torgleich mit der vorherigen */

   Tabellenplatz  = CurRang;

   /* Nicht benötigte Variablen loswerden */
   DROP  CurRang CurTeam;

RUN;
```

4.9 Subset einer Tabelle in einem WORK-Dataset speichern

- Szenario
  - Die Datenaufbereitung ist abgeschlossen und es kann mit den Auswertungen können begonnen werden
  - Für die geplanten Auswertungen sollen Hilfsdateien mit Daten einer Saison und eines bestimmten Teams in dieser Saison erzeugt werden
    - LIGA.LIGATABELLE und LIGA.ERGEBNISSE müssen gefiltert werden
  - Die Aufgabe lautet demnach: Einfaches Filtern von Datasets ohne zusätzliche Manipulationen

- Zur Verfügung stehende Techniken
  - PROC SQL mit CREATE TABLE, SELECT FROM und WHERE-Statement
  - DATA Step mit SET und WHERE-Statement

- Empfehlung für/ Vorteile der anzuwendende(n) Technik
  - Keine der Varianten hat herausragende Vor- oder Nachteile
- TIPP
  - Wenn das Ergebnis sortiert sein soll, ist PROC SQL im Vorteil, da kein separater Sortierschritt notwendig ist.
- Spielstand nach diesem Arbeitsschritt
  - Es bleibt beim 4 : 3 für den DATA Step
- Beispielprogramm

```
%LET p_sSaison    = 2009/10;
%LET p_sTeam      = SC Freiburg;

/**
   Abschließend werden Staistiken für eine Saison bzw. für
   ein bestimmtes Team in dieser Saison aufbereitet.
   -  Die Ergebnistabelle und die Ligatabelle müssen für die
      Auswertung entsprechend gefiltert werden.
   -  Saison und Team werden als Macrovariablen gesetzt. So
      lässt sich schnell eine andere Auswertung durchführen.
/**/
/**   Alternative 1: PROC SQL
/**/
PROC SQL;
   CREATE TABLE WORK.ltSaison AS
   SELECT      *
      FROM     LIGA.LIGATABELLE
      WHERE    Saison      EQ "&p_sSaison."
      AND      OK          EQ 1
      ORDER BY Spieltag
             ,  TabellenPlatz
   ;
QUIT;


PROC SQL;
   CREATE TABLE WORK.ltSaisonTeam AS
   SELECT      *
      FROM     WORK.ltSaison
      WHERE    Team     EQ "&p_sTeam."
      ORDER BY Saison
             ,  Spieltag
   ;
QUIT;


/**   Alternative 2: DATA Step
/**/
DATA WORK.resSaison;
   SET LIGA.RESULTS;
   WHERE (Saison EQ "&p_sSaison.");
RUN;
```

```
DATA WORK.resSaisonTeam;
   SET WORK.resSaison;
   WHERE    HeimVerein  EQ "&p_sTeam."
   OR       GastVerein  EQ "&p_sTeam."
   ;
RUN;

PROC SORT
   DATA  =  WORK.resSaison;
   BY       Spieltag
            SpielAmSpieltag
   ;
RUN;
PROC SORT
   DATA  =  WORK.resSaisonTeam;
   BY       Spieltag
            SpielAmSpieltag
   ;
RUN;
```

4.10 Einzelstatistiken ermitteln und Ergebnisse in Makrovariablen speichern

- Szenario
  - Einzelfragen sollen beantwortet werden können
    - Welches Team hat die meisten Tore in einem Spiel geschossen?
    - In welchem Spiel fielen die meisten Tore?
    - Hat der Spielbeginn (vor oder nach 19:00) einen Einfluss auf die Heimspiel-Bilanz?
    - …
- Zur Verfügung stehende Techniken
  - PROC SQL mit SELECT INTO-Statement
  - DATA Step mit CALL SYMPUTX-Routine
- Empfehlung für/ Vorteile der anzuwendende(n) Technik
  - Beide Varianten haben hier ihre Vorteile; kein Sieger
  - Vorteile PROC SQL
    - Einfache Schreibweise
    - Mehrere Werte als Zeichenkette oder als Makro-Variablenlisten abrufbar
  - Vorteile DATA Step
    - Mittels BY-Verarbeitung können mehrere Aggregationsstufen gleichzeitig ausgewertet werden
- Spielstand nach diesem Arbeitsschritt
  - Keine Ergebnisänderung mehr. Es bleibt beim 4 : 3 für den DATA Step

- Beispielprogramm

```
/**
   Einzelstatistiken (z.B. wer hat wann die meisten Tore in
   der gewählten Saison geschossen) werden erzeugt
/**/
/**   Alternative 1: PROC SQL
/**/
/* Wer hat wann die meisten Tore geschossen?
   (mta=Max. Tore Absolut) */
PROC SQL NOPRINT;
   SELECT      "Am meisten erzielte Tore"
             , Team
             , Spieltag
             , Datum
             , TorePlusHeute
      INTO     :mta_Descr1          - :mta_Descr18
             , :mta_Team1           - :mta_Team18
             , :mta_Spieltag1       - :mta_Spieltag18
             , :mta_Datum1          - :mta_Datum18
             , :mta_MaxTore1        - :mta_MaxTore18
      FROM     ltSaison
      HAVING   TorePlusHeute EQ Max (TorePlusHeute)
   ;
QUIT;
%LET mta_Count  = &SQLOBS.;

/* In welchem Spiel sind die meisten Tore gefallen?
   (mtsp=Max. Tore in einem Spiel) */
PROC SQL NOPRINT;
   SELECT      "Spiel mit den meisten Toren"
             , HeimVerein
             , GastVerein
             , Spieltag
             , Datum
             , ToreHeim
             , ToreGast
             , ToreHeim + ToreGast     AS ToreGesamt
      INTO     :mtsp_Descr1         - :mtsp_Descr18
             , :mtsp_Heim1          - :mtsp_Heim18
             , :mtsp_Gast1          - :mtsp_Gast18
             , :mtsp_Spieltag1      - :mtsp_Spieltag18
             , :mtsp_Datum1         - :mtsp_Datum18
             , :mtsp_ToreHeim1      - :mtsp_ToreHeim18
             , :mtsp_ToreGast1      - :mtsp_ToreGast18
             , :mtsp_MaxTore1       - :mtsp_MaxTore18
      FROM     resSaison
      HAVING   (ToreHeim+ToreGast) EQ Max (ToreHeim+ToreGast)
   ;
QUIT;
%LET mtsp_Count  = &SQLOBS.;
```

```
/* Dataset erzeugen, in dem die Einzelstatistiken gespeichert
   werden. Dieses Dataset kann dann wiederum mit Report-
   Prozeduren ausgewertet werden.
*/
DATA Liga.Einzelstatistik;
   Length
      Anzahl                  8
      i                       8
      MacVar                  $50
      Statistik               $100
      SpielTag                4
      Datum                   8
      Team1                   $50
      Team2                   $50
      ToreTeam1               4
      ToreTeam2               4
      ToreGesamt              4
   ;
   Format
      Datum                   ddmmyy10.
   ;

   Anzahl          = SymgetN ('mta_Count');
   DO i = 1 TO Anzahl;
      MacVar          = CatS ('mta_Descr', i);
      Statistik       = Symget (MacVar);
      MacVar          = CatS ('mta_Spieltag', i);
      SpielTag        = SymgetN (MacVar);
      MacVar          = CatS ('mta_Datum', i);
      Datum           = Input (Symget (MacVar), ddmmyy10.);
      MacVar          = CatS ('mta_Team', i);
      Team1           = Symget (MacVar);
      MacVar          = CatS ('mta_MaxTore', i);
      ToreTeam1       = SymgetN (MacVar);
      ToreGesamt      = ToreTeam1;
      OUTPUT;
   END; /* Über alle Teams,. die die meisten Tore
           geschossen haben */

   Anzahl          = SymgetN ('mtsp_Count');
   DO i = 1 TO Anzahl;
      MacVar          = CatS ('mtsp_Descr', i);
      Statistik       = Symget (MacVar);
      MacVar          = CatS ('mtsp_Spieltag', i);
      SpielTag        = SymgetN (MacVar);
      MacVar          = CatS ('mtsp_Datum', i);
      Datum           = Input (Symget (MacVar), ddmmyy10.);
      MacVar          = CatS ('mtsp_Heim', i);
      Team1           = Symget (MacVar);
      MacVar          = CatS ('mtsp_Gast', i);
      Team2           = Symget (MacVar);
      MacVar          = CatS ('mtsp_ToreHeim', i);
```

```
      ToreTeam1      = SymgetN (MacVar);
      MacVar         = CatS ('mtsp_ToreGast', i);
      ToreTeam2      = SymgetN (MacVar);
      ToreGesamt     = Sum (ToreTeam1, ToreTeam2);
      OUTPUT;
   END; /* Über alle Begegnungen, in denen die meisten Tore
           insgesamt fielen */

   DROP i Anzahl MacVar;
RUN;


/**
   Einzelstatistiken (z.B. hat der Spielbeginn einen Einfluss
   auf die Heimbilanz?)
/**/
/**   Alternative 1: PROC SQL, um einzelne Statistiken in
                     Macrovariablen zu speichern und
                     anschließend DATA Step, wo diese Werte
                     wiederverwendet werden
/**/
/* Zunächst bestimmte Statistiken ermitteln */
PROC SQL NOPRINT;
   SELECT      Count (HeimVerein)
      INTO     :g_iHeimSiege
      FROM     resSaison
      WHERE    SiegHeim          EQ 1
   ;
   SELECT      Count (HeimVerein)
      INTO     :g_iHeimRemis
      FROM     resSaison
      WHERE    RemisHeim         EQ 1
   ;
   SELECT      Count (HeimVerein)
      INTO     :g_iHeimNiederlage
      FROM     resSaison
      WHERE    NiederlageHeim    EQ 1
   ;
QUIT;
/* Zusätzliche Werte mit MACRO - Programmierung oder
   'DATA _NULL_ Step' ermitteln und nicht erneut selektieren
   -> Performance-Aspekt
*/
%LET g_iAnzahlSpiele       = %EVAL (&g_iHeimSiege.
                                    + &g_iHeimRemis.
                                    + &g_iHeimNiederlage.
                                   );
%LET g_iGastSiege          = &g_iHeimNiederlage.;
%LET g_iGastRemis          = &g_iHeimRemis.;
%LET g_iGastNiederlage     = &g_iHeimSiege.;
DATA _NULL_;
   iAnzahlSpiele           = SymgetN ('g_iAnzahlSpiele');
```

```
   iHeimSiege              = SymgetN ('g_iHeimSiege');
   iHeimRemis              = SymgetN ('g_iHeimRemis');
   iHeimNiederlage         = SymgetN ('g_iHeimNiederlage');

   nQuoteHS = Round (iHeimSiege      / iAnzahlSpiele, 0.01);
   nQuoteHR = Round (iHeimRemis      / iAnzahlSpiele, 0.01);
   nQuoteHN = Round (iHeimNiederlage / iAnzahlSpiele, 0.01);

   Call SymputX ('g_nQuoteHeimSieg',          nQuoteHS);
   Call SymputX ('g_nQuoteHeimRemis',         nQuoteHR);
   Call SymputX ('g_nQuoteHeimNiederlage',    nQuoteHN);
RUN;
%PUT g_nQuoteHeimSieg: &g_nQuoteHeimSieg.;

/* Zwischendatei mit weiteren Zwischenergebnissen erzeugen */
PROC SQL NOPRINT;
   CREATE TABLE AbendSpiele AS
      SELECT   AbendSpiel
             , Count (HeimVerein)       AS AnzSpiele
             , Sum (ToreHeim)           AS ToreHeim
             , Sum (ToreGast)           AS ToreGast
             , Sum (HalbzeitHeim)       AS HalbzeitHeim
             , Sum (HalbzeitGast)       AS HalbzeitGast
             , Sum (PunkteHeim)         AS PunkteHeim
             , Sum (PunkteGast)         AS PunkteGast
             , Sum (HalbzeitPunkteHeim) AS HalbzeitPunkteHeim
             , Sum (HalbzeitPunkteGast) AS HalbzeitPunkteGast
             , Sum (SiegHeim)           AS SiegHeim
             , Sum (RemisHeim)          AS RemisHeim
             , Sum (NiederlageHeim)     AS NiederlageHeim
             , Sum (SiegGast)           AS SiegGast
             , Sum (RemisGast)          AS RemisGast
             , Sum (NiederlageGast)     AS NiederlageGast
      FROM     resSaison
      GROUP BY AbendSpiel
      ORDER BY AbendSpiel
   ;
QUIT;

/* Aufbereiten eines Datasets, das die Heimsieg-Quoten in
   Abhängigkeit des Spielbeginns ausweist
   Dieses Dataset kann dann wiederum mit Report-Prozeduren
   ausgewertet werden.
*/
DATA Liga.AbendSpielStatistik;
   SET AbendSpiele;

   Length
      Typ            $30
   ;

   Typ      = "01 - HeimSieg";
```

```
    Quote     = Round (SiegHeim          / AnzSpiele, 0.01);
    GesQuote = &g_nQuoteHeimSieg.;
    Output;

    Typ       = "02 - Remis";
    Quote     = Round (RemisHeim         / AnzSpiele, 0.01);
    GesQuote = &g_nQuoteHeimRemis.;
    Output;

    Typ       = "03 - HeimNiederlage";
    Quote     = Round (NiederlageHeim    / AnzSpiele, 0.01);
    GesQuote = &g_nQuoteHeimNiederlage.;
    Output;

    KEEP
       AbendSpiel
       Typ
       Quote
       GesQuote
    ;
RUN;
```

5 Abschließende Betrachtungen

Das Tutoriums-Projekt hat gezeigt, dass es keinen eindeutigen Favoriten zwischen den beiden grundlegenden Methoden der Datenmanipulation in SAS gibt.
Es gibt genügend Szenarien, wo es eigentlich keine Rolle spielt, welches Verfahren man anwendet. Andererseits hängt es immer vom Einzelfall ab, ob eine Variante im Vorteil ist.
Ein möglicher Aspekt im Vergleich der beiden Methoden könnte der Performanceaspekt sein. Hat der DATA Step Performancevorteile gegenüber dem PROC SQL oder umgekehrt?
Leider bringt auch dieses Thema keine Wende. Wenn Performance relevant ist, muss man wiederum im Einzelfall prüfen, wie vorzugehen ist.

- In Bezug auf Performance ist sicher eine der wichtigsten Regeln, dass die Anzahl der Verarbeitungsschritte so weit wie möglich reduziert werden sollte, unabhängig davon, ob es sich um DATA Steps oder PROC SQLs handelt.

Abschließend werden in den folgenden beiden Abschnitten die Vorteile bzw. Alleinstellungsmerkmale der beiden Verfahren aufgelistet.

5.1 Vorteile/Alleinstellungsmerkmale für DATA Step

- Einlesen von Rohdaten aus Textdateien
- Einlesen von mehreren Rohdaten/Datasets/Tabellen in einem Step möglich
  - Sequentieller als auch direkter Zugriff (KEY=/POINT=) mittels SET-Statement möglich
- Erzeugen mehrerer Ausgabe-Datasets in einem Step möglich

- Bei MERGE können gleichzeitig die „Volltreffer“, die „linken“ und die „rechten“ Treffer separiert werden
- Komplette Mächtigkeit einer (prozeduralen) Programmiersprache im DATA Step
    - Ablaufsteuerung (IF/THEN/ELSE; DO WHILE/UNTIL; …)
    - Spezielle DATA Step – Funktionen
    - Objektorientierte Erweiterungen
        - HASH-Tabellen – Objekte
        - PRX-Funktionen
- BY-Verarbeitung für sehr flexible Steuerung (z.B. bei Aggregationen)
    - Erfordert vorherige Sortierung bzw. Indexierung!

5.2 Vorteile/Alleinstellungsmerkmale für PROC SQL

- Bilden eines Kartesischen Produkts der Zeilen zweier Datasets
- Nicht nur EQUI-JOINS sind möglich
    - Z.B.: WHERE a.X GT b.Y
- Datasets müssen vor einem JOIN nicht sortiert sein
- Sortieren der Ergebnismenge im gleichen Schritt möglich
- Einfaches Aggregieren (incl. Bedingungsfestlegung für aggregierte Werte)
    - GROUP BY (in Kombination mit HAVING)
- In einem Step können Gruppenergebnisse in Relation zu Gesamtergebnissen gesetzt werden
    - Anteilsberechnung (Stichwort: REMERGING …)
- Sehr einfaches Abspeichern von Werten in Macrovariablen