Geschwindigkeit ist nicht alles – wozu Hash-Objekte wirklich gut sind

Arne Leißner
Entimo AG
Stralauer Platz 33-34
10243 Berlin
arne.leissner@entimo.de

Zusammenfassung

Schlägt man im KSFE-Vortragsarchiv nach, so finden sich dort vorerst nur wenige Beiträge, die auf das SAS Hash-Objekt eingehen. Darin wird das Hash-Objekt als eine weitere Lookup und Join-Technik vorgestellt, als Alternative zu PROC SQL, als Performancebeschleuniger und als Beispiel einer gelungenen Objektintegration in den SAS Data Step. Das als Standardbeispiel gewählte Einlesen einer kompletten SAS-Datei in ein Hash-Objekt vermittelt allerdings nur ein unvollständiges Bild darüber, was Hash-Objekte tatsächlich können und welches Einsatzpotential sie in sich bergen.
Einige dieser Möglichkeiten werden an eingängigen Beispielen demonstriert.

Schlüsselwörter: DataStep, Hash-Objekt

1 Einführung

Das Hash-Objekt stellt neben dem Program Data Vector (PDV) und den LAG-Queues eine weiteren Form einer im Data Step verwendbaren Datenstruktur bereit. Allen gemein ist, dass diese Datenstrukturen im Hauptspeicher angelegt werden und nur für die Dauer des Data Steps existieren, also flüchtig sind.
Im Hash-Objekt sind die Einträge über Schlüssel adressiert. Die spezifischen Eigenschaften des Hash-Objekts werden im direkten Vergleich mit dem PDV deutlich:

Tabelle 1: Vergleich Program Data Vector und Hash-Objekt

Program Data Vector	Hash-Objekt
Eindimensionales Tupel	Zweidimensionale Datenstruktur
Systemkontrolliert	Individuell kontrolliert
Immer vorhanden	Dann und solange, wie man möchte
Gibt es genau einmal	So viele, wie man möchte
Zur Compile Time erzeugt	Zur Execution Time erzeugt
Struktur: Alle Data Step Variablen	Struktur frei wählbar
Elemente gleichberechtigt	Schlüssel und Datenelemente

Genau 1 Eintrag = 1 Beobachtung	Beliebig viele Einträge
I.d.R. je Iteration neu initialisiert	Iterationsinvariant

Die nachfolgende Programmskizze demonstriert diese Eigenschaften in der praktischen Verwendung:

```
DATA lab_pat;
   SET lab;
   IF _N_ = 1 THEN DO;
      IF flag THEN dsn = "patinfo_enh";
      ELSE dsn = "patinfo";
      DCL HASH h (DATASET: dsn);
      h.DEFINEKEY("subjid");
      h.DEFINEDATA("age", "sex");
      IF flag THEN h.DEFINEDATA("weight", "height");
      h.DEFINEDONE();
   END; /* _N_ = 1 */
...
   IF  measure > 75 THEN rc = h.FIND();
...
RUN;
```

1. Das Hash-Objekt wird bedingungsabhängig (IF _N_ = 1) zur Execution Time angelegt.
2. Ohne Bedingung würde bei jeder Data-Step-Iteration unnötigerweise ein neues Hash-Objekt angelegt und gefüllt werden (also so oft und so viele, wie hier die SAS-Datei *lab* Sätze hätte).
3. Die Datenquelle für das Hash-Objekt *h* wird datenabhängig erst zur Execution Time festgelegt (entsprechend dem Inhalt der Variablen *flag* aus der ersten Beobachtung in der SAS-Datei *lab*).
4. Minimal muss ein Schlüsselattribut mittels DEFINEKEY festgelegt werden.
5. Die Struktur des Hash-Objekts kann schrittweise und bedingungsabhängig definiert werden (DEFINEDATA)
6. Nach der Definition des Hash-Objekts beinhaltet dieses
   - 1Schlüsselspalte,
   - 2 oder 4 Datenspalten
   - so viele Einträge wie Sätze in der Quell-Datei (*pat_info* oder *patinfo_enh*)
7. Auch wenn bei _N_ = 1 angelegt: Hash-Objekt ist in jeder Iteration über den Identifier *h* referenzierbar
8. Für die Schlüsselsuche im Hash-Objekt (FIND) wird automatisch der Inhalt der Variablen *subjid* aus der aktuellen Beobachtung der SAS-Datei *lab* verwendet. Wurde die *subjid* im Hash-Objekt gefunden, stehen die im Hash-Objekt dazu gespeicherten Datenwerte in den namensgleichen Variablen des PDV's (*age*, *sex* sowie ggf. *weight* und *height*) zur Verfügung.

Tabelle 2: Benefits des Hash-Objektes

Benefits
Zusätzliche dynamische Datenstruktur(en) im Direktzugriff des DATA Steps
Einträge über Schlüsselwerte indiziert und auffindbar (Speicherposition = Bucket via Hashkey, Ablage erfolgt unsortiert)
Simple und composite Keys unterstützt (numerisch, alphanumerisch)
Sehr schnelle Suchalgorithmen unabhängig von der Anzahl der Einträge (Hashfunktionen in Memory)
Können und dürfen groß werden
Freie Navigation (via Schlüssel od. sequentiell) -> Direktadressierung -> Multiadressierung -> schrittweise vorwärts / rückwärts mittels Iterator
Dynamische und agile Techniken für das anlegen, modifizieren und löschen von Einträgen (manuell und automatisch)
Verlinkung zu Data Step Variablen über den Namen -> Automatischer Informationsaustausch zwischen PDV und Hash-Objekt (wahlweise auch manuelle Zuweisungen)
Einfacher Import / Export von SAS Data Sets mit Data Set Optionen
Eingebaute Summenfunktion
Steuerung des Verhaltens via attribute tags
Objektorientierter Ansatz (Methoden, Attribute, Dot-Notation)

2 Beispiele für Techniken und Use Cases

Das Hash-Objekt unterstützt eine Vielzahl von Anwendungsszenarien und erweitert die Menge der technischen Optionen zur Lösung ausgewählter Aufgabenstellungen

Tabelle 3: Einsatzmöglichkeiten für das Hash-Objekt (Auswahl)

Techniken & Use Cases
Lookop
Sortierung
De-Duplizierung
Zählung
Summierung
Durchwanderung
Data Set Splitting
Mehrfachlesen
Fuzzy Merge
Top-<n>%
Gruppenstatistiken
Join
Rekursives Lookup
Relationsprüfungen
Compare
Update
…

Für die in Tabelle 2 unterstrichenen Anwendungsszenarien finden sich in der einführenden Programmskizze und im nachfolgenden Kapitel einfache und aufeinander aufbauende Prinzipbeispiele (jeweils mit Aufgabenstellung, Lösungsidee, SAS-Programm). Sie sind so angelegt, dass keine langen Erklärungen zu dem Anliegen und den verwendeten Inputdaten erforderlich sind und dass die zur Illustration des Lösungsweges abgebildeten SAS-Programme in jedem Fall nur wenige Zeilen umfassen.

3 Beispiele

3.1 Sortierung

Aufgabe	SAS-Datei *sashelp.class* soll sortiert nach Alter und Geschlecht ausgegeben werden.
Verwendete Features	attribute-tags: DATASET, ORDERED, MULTIDATA Methode: OUTPUT
Lösungsidee	Durch ORDERED: „A“ erfolgt automatisch eine anhand der Schlüsselattribute aufsteigend (ascending) sortierte Dateiausgabe (OUTPUT-METHODE). Mit MULTIDATA: „Y“ wird das Hash-Objekt angewiesen, mehrfache Dateneinträge zu ein und demselben Schlüssel zu akzeptieren (Hintergrund: In *sashelp.class* sind je Kombination aus Alter und Geschlecht bis zu drei Sätze vorhanden).

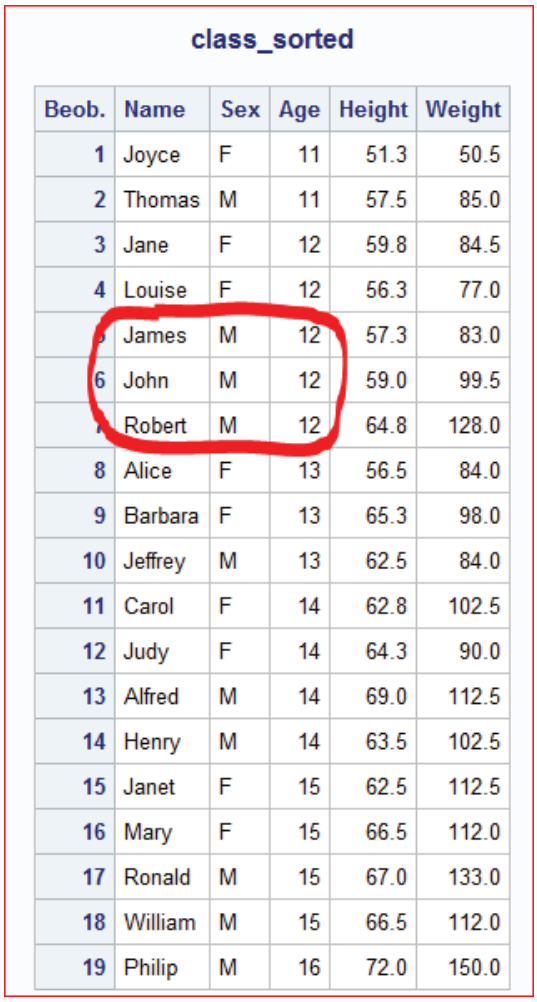
Programmbeispiel:

```
DATA _null_;
   IF 0 THEN SET sashelp.class;
   DCL HASH h(DATASET: "sashelp.class",
   ORDERED: "A", MULTIDATA: "Y");
   h.DEFINEKEY("age", "sex");
   h.DEFINEDATA(ALL: "Y");
   h.DEFINEDONE();
   h.OUTPUT(DATASET "work.class_sorted");
RUN;

NOTE: There were 19 observations read from the data set
      SASHELP.CLASS.
NOTE: The data set WORK.CLASS_SORTED has 19 observations and
      5 variables.
```

Es werden 19 Sätze eingelesen und 19 Sätze ausgegeben. Die Ausgabedatei enthält zu einzelnen Kombinationen aus Alter und Geschlecht mehrere Beobachtungen (z.B. James, John und Robert bei den 12jährigen Jungen).

Das Konstrukt IF 0 THEN SET … bewirkt, dass die Eigenschaften (Typ und Länge) für die Datenspalten *age* und *sex* aus *sashelp.class* ermittelt und dem Hash-Objekt zur Verfügung gestellt werden, ohne diese Datei mittels SET-Anweisung zu lesen.



class_sorted

Beob.	Name	Sex	Age	Height	Weight
1	Joyce	F	11	51.3	50.5
2	Thomas	M	11	57.5	85.0
3	Jane	F	12	59.8	84.5
4	Louise	F	12	56.3	77.0
5	James	M	12	57.3	83.0
6	John	M	12	59.0	99.5
7	Robert	M	12	64.8	128.0
8	Alice	F	13	56.5	84.0
9	Barbara	F	13	65.3	98.0
10	Jeffrey	M	13	62.5	84.0
11	Carol	F	14	62.8	102.5
12	Judy	F	14	64.3	90.0
13	Alfred	M	14	69.0	112.5
14	Henry	M	14	63.5	102.5
15	Janet	F	15	62.5	112.5
16	Mary	F	15	66.5	112.0
17	Ronald	M	15	67.0	133.0
18	William	M	15	66.5	112.0
19	Philip	M	16	72.0	150.0

3.2 De-Duplizierung (1)

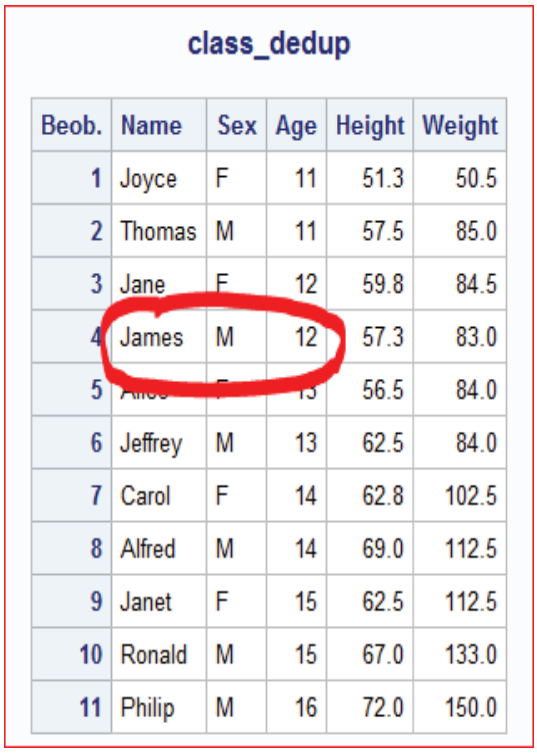
Aufgabe	Anknüpfend an das vorangegangene Beispiel soll je Kombination aus Alter und Geschlecht nur ein Satz ausgegeben werden.
Verwendete Features	attribute-tags: MULTIDATA: “N”
Lösungsidee	Mit MULTIDATA: „N“ wird das Hash-Objekt angewiesen, je Schlüsselwert nur einen Eintrag zuzulassen

Programmbeispiel:

```
DATA _null_;
   IF 0 THEN SET sashelp.class;
   DCL HASH h(DATASET: "sashelp.class",
              ORDERED: "A", MULTIDATA: "N");
   h.DEFINEKEY("age", "sex");
   h.DEFINEDATA(ALL: "Y");
   h.DEFINEDONE();
   h.OUTPUT(DATASET "work.class_sorted");
RUN;
```

```
NOTE: There were 19 observations read from the data set
      SASHELP.CLASS.
NOTE: The data set WORK.CLASS_SORTED has 11 observations and
      5 variables.
```

Es werden 19 Sätze eingelesen und 11 Sätze (je ein Satz pro vorgefundener Kombination aus Alter und Geschlecht) ausgegeben. Bei den 12jährigen Jungen ist nur noch James enthalten. MULTIDATA: „N“ bewirkt, dass nur der erste Satz zu einem gewählten Schlüssel im Hash-Objekt gespeichert wird und alle nachfolgenden Sätze mit gleichem Schlüssel ignoriert werden.



class_dedup

Beob.	Name	Sex	Age	Height	Weight
1	Joyce	F	11	51.3	50.5
2	Thomas	M	11	57.5	85.0
3	Jane	F	12	59.8	84.5
4	James	M	12	57.3	83.0
5	[illegible]	F	13	56.5	84.0
6	Jeffrey	M	13	62.5	84.0
7	Carol	F	14	62.8	102.5
8	Alfred	M	14	69.0	112.5
9	Janet	F	15	62.5	112.5
10	Ronald	M	15	67.0	133.0
11	Philip	M	16	72.0	150.0

3.3 De-Duplizierung (2)

Aufgabe	Anknüpfend an das vorangegangene Beispiel soll je Kombination aus Alter und Geschlecht nur der letzte Satz ausgegeben werden.
Verwendete Features	attribute-tag: DUPLICATE: “R”
Lösungsidee	Mit DUPLICATE: „R“ (Replace) wird das Hash-Objekt angewiesen, bei MULTIDATA: „N“ (Default) je Schlüsselwert nur einen Eintrag zuzulassen und bei Mehrfachvorkommen den vorhandenen Eintrag mit den jeweils neuen Daten zu diesem Schlüssel zu überschreiben.

Programmbeispiel:

```
DATA _null_;
   IF 0 THEN SET sashelp.class;
   DCL HASH h(DATASET: "sashelp.class",
              ORDERED: "A", DUPLICATE: "R");
   h.DEFINEKEY("age", "sex");
   h.DEFINEDATA(ALL: "Y");
   h.DEFINEDONE();
   h.OUTPUT(DATASET "work.class_sorted");
RUN;

NOTE: There were 19 observations read from the data set
      SASHELP.CLASS.
NOTE: The data set WORK.CLASS_SORTED has 11 observations and
      5 variables.
```

Bei den 12jährigen Jungen wird der letzte Satz mit Robert die vorherigen im Hash Objekt.

class_dedup

Beob.	Name	Sex	Age	Height	Weight
1	Joyce	F	11	51.3	50.5
2	Thomas	M	11	57.5	85.0
3	Louise	F	12	56.3	77.0
4	Robert	M	12	64.8	128.0
5	Barbara	F	13	65.3	98.0
6	Jeffrey	M	13	62.5	84.0
7	Judy	F	14	64.3	90.0
8	Henry	M	14	63.5	102.5
9	Mary	F	15	66.5	112.0
10	William	M	15	66.5	112.0
11	Philip	M	16	72.0	150.0

3.4 Zählung

Aufgabe	Es soll die Zahl der Sätze je Kombination aus Alter und Geschlecht in *sashelp.class* gezählt werden.
Verwendete Features	attribute-tag: SUMINC: <“incrementvariable“> Methoden SUM und REF
Lösungsidee	Mit SUMINC: „<incrementvariable>“ wird zu jedem im Hash-Objekt gespeicherten Schlüssel ein zusätzlicher Summary Value angelegt. Jede Prüfung auf die Existenz eines Schlüssels im Hash-Objekt führt dazu, dass der Summary Value um den aktuellen Wert der Inkrement-variablen erhöht wird.

Programmbeispiel:

```
DATA _null_;
   DCL HASH h(SUMINC: "increment");
   h.DEFINEKEY("age", "sex");
   h.DEFINEDONE();
   increment = 1;
   DO UNTIL (eof);
      SET sashelp.class(keep = age sex) END = eof;
      rc = h.REF();     /* REF() = CHECK() + ADD() */
      rc = h.SUM(SUM: count);
      satz + 1;
      PUT @5 satz= @15 age= sex= count=;
   END;
RUN;

NOTE: There were 19 observations read from the data set
      SASHELP.CLASS.
```

In diesem Programmbeispiel wird das Hash-Objekt satzweise innerhalb einer Einleseschleife gefüllt. Da der Wert der Inkrementvariablen *increment* auf 1 gesetzt wird, wirkt sie wie ein einfacher Zähler bei jeder Ausführung der REF-METHODE (stellt eine Zusammenfassung der CHECK-Methode und der ADD-Methode dar). Die kontinuierliche Veränderung der Summary Values (Rückgabe in die Variable *count* bei Aufruf der SUM-Methode) ist im SAS-Log dargestellt.

```
satz=1    Age=14 Sex=M count=1
satz=2    Age=13 Sex=F count=1
satz=3    Age=13 Sex=F count=2
satz=4    Age=14 Sex=F count=1
satz=5    Age=14 Sex=M count=2
satz=6    Age=12 Sex=M count=1
satz=7    Age=12 Sex=F count=1
satz=8    Age=15 Sex=F count=1
satz=9    Age=13 Sex=M count=1
satz=10   Age=12 Sex=M count=2
satz=11   Age=11 Sex=F count=1
satz=12   Age=14 Sex=F count=2
satz=13   Age=12 Sex=F count=2
satz=14   Age=15 Sex=F count=2
satz=15   Age=16 Sex=M count=1
satz=16   Age=12 Sex=M count=3
satz=17   Age=15 Sex=M count=1
satz=18   Age=11 Sex=M count=1
satz=19   Age=15 Sex=M count=2
```

3.5 Summierung

Aufgabe	Es sollen die zu drei Produkten „A“, „B“ und „C“ für die an Januar 2010 verfügbaren Monatsumsätze die Jahresumsätze ermittelt und in eine neue SAS-Datei ausgegeben werden.
Verwendete Features	attribute-tag: SUMINC: <“incrementvariable“> und KEYSUM “sum-variable”

	Methoden FIND und OUTPUT
Lösungsidee	Anders als im Beispiel 3.4 wird mittels Attribute Tag SUMKEY für den Summary Value ein Name vergeben. Bei der Ausgabe in eine Datei wird der SUMKEY als zusätzliche Spalte mit ausgegeben. Die Inkrementvariable muss keinen festen Wert besitzen, sondern kann veränderlich (und sogar negativ) sein. Das Lösungsbeispiel nutzt dieses Feature dahingehend aus, als dass die Monatsumsätze aus der Eingabedatei als Inkrement verwendet werden.

Programmbeispiel:

```
DATA _null_;
   IF 0 THEN SET monatsumsatz;
   DCL HASH h_sum(SUMINC: "umsatz", KEYSUM: "umsatz",
                  ORDERED: "A");
   h_sum.DEFINEKEY("produkt", "jahr");
   h_sum.DEFINEDATA("produkt", "jahr");
   h_sum.DEFINEDONE();

   DO UNTIL (eof);
      SET monatsumsatz (keep = jahr produkt umsatz) end = eof;
      IF h_sum.FIND() THEN h_sum.ADD();
   END;
   h_sum.OUTPUT(DATASET: "jahresumsatz");
RUN;

NOTE: There were 180 observations read from the data set
      WORK.MONATSUMSATZ.
NOTE: The data set WORK.JAHRESUMSATZ has 15 observations and 3
      variables.
```

Als Schlüsselattribute für das Hash-Objekt werden *produkt* und *jahr* verwendet. Bei jedem Eingabesatz mit dem Monatsumsatz zu einem Produkt wird der Jahreswert um den Monatswert erhöht. Die Variable umsatz ist automatisch Bestandteil der mittels OUTPUT-Methode erzeugten Ergebnisdatei.

Ein Ausschnitt der Eingabedatei und die Ausgabedatei sind nachfolgend dargestellt.

work.monatsumsatz (obs = 16)

Beob.	produkt	jahr	monat	umsatz
1	A	2010	1	€1.644,7
2	B	2010	1	€486,2
3	C	2010	1	€941,9
4	A	2010	2	€1.809,4
5	B	2010	2	€487,5
6	C	2010	2	€994,9
7	A	2010	3	€1.762,7
8	B	2010	3	€493,0
9	C	2010	3	€1.073,5
10	A	2010	4	€1.773,3
11	B	2010	4	€542,2
12	C	2010	4	€1.047,1
13	A	2010	5	€1.914,5
14	B	2010	5	€567,8
15	C	2010	5	€1.129,1
16	A	2010	6	€1.984,4

work.jahresumsatz

Beob.	produkt	jahr	umsatz
1	A	2010	€269.433,5
2	A	2011	€282.708,8
3	A	2012	€300.262,3
4	A	2013	€308.316,1
5	A	2014	€325.509,2
6	B	2010	€62.823,6
7	B	2011	€65.066,0
8	B	2012	€67.258,4
9	B	2013	€70.900,6
10	B	2014	€73.997,5
11	C	2010	€155.348,9
12	C	2011	€164.830,6
13	C	2012	€172.118,2
14	C	2013	€176.715,6
15	C	2014	€190.105,5

3.6 Data Set Split, Durchwanderung, Mehrfachlesen

Aufgabe	Aus sashelp.class soll je Alter und Geschlecht eine neue SAS-Datei erzeugt werden, ohne vorher Zahl und Inhalt der Kombinationen zu kennen.
Verwendete Features	Mehrere Hash-Objekte Mehrfachlesen einer Datei Hash-Iterator Methoden FIRST, NEXT und OUTPUT Data Set Optionen
Lösungsidee	Ein erstes Hash-Objekt *h* enthält *sashelp.class* vollständig. Es bildet somit die Datenquelle für die Ausgabe in die zu erzeugenden Teildateien. Ein zweites Hash-Objekt *h_key* beinhaltet nur die Schlüsselkombinaten aus *sashelp.class* und steuert darüber die Erzeugung der Ausgabedateien. Das zweite Hash-Objekt wird mit einem Hash-Iterator *h_key_iter* versehen, so dass es über die verschiedenen Schlüssel „durchwandert“ werden kann.

Programmbeispiel:

```
DATA _null_;
   IF 0 THEN SET sashelp.class;
   DCL HASH h(DATASET: "sashelp.class", MULTIDATA: "Y");
   h.DEFINEKEY("age", "sex");
   h.DEFINEDATA(ALL: "Y");
   h.DEFINEDONE();
   DCL HASH h_key(DATASET: "sashelp.class(keep = age sex)");
   h_key.DEFINEKEY("age", "sex");
   h_key.DEFINEDONE();
   DCL HITER h_key_iter("h_key"); /* Iterator */
   rc = h_key_iter.FIRST();
   DO UNTIL(h_key_iter.NEXT());
      h.OUTPUT(DATASET: "class_"!!strip(sex)!!"_" !!
                        strip(put(age, 3.)) !!
                        "(WHERE = (sex='"!!strip(sex) !!
                        "' and age=" !!
                        strip(put(age, 3.))!!"))");
   END;
RUN;

NOTE: There were 19 observations read from the data set
      SASHELP.CLASS.
NOTE: There were 19 observations read from the data set
      SASHELP.CLASS.
NOTE: The data set WORK.CLASS_F_11 has 1 observations and
      5 variables.
NOTE: The data set WORK.CLASS_M_11 has 1 observations
      and 5 variables.
NOTE: The data set WORK.CLASS_F_12 has 2 observations and
      5 variables
```

Sashelp.class wird zweimal in unterschiedliche Hash-Objekte eingelesen. Mittels Iterator-Methode FIRST wird anschließend zunächst der erste Schlüssel angesprochen und mittels wiederholtem NEXT werden die Schlüsselwerte nacheinander durchlaufen (Sortierreihenfolge ist an dieser Stelle beliebig). Bei der Erzeugung der Ausgabedateien wird ausgenutzt, dass der Dateiname für die OUTPUT-Methode erst zu Laufzeit gebildet und zugewiesen werden kann. Durch Anwendung einer WHERE-Klausel als Data Set Option für die Ausgabedateien, wird der jeweils entsprechende Filter gesetzt. Die Inhalte der hierfür verwendeten Variablen *age* und *sex* werden automatisch durch FIRST und NEXT gesetzt.

3.7 Fuzzy Merge

Aufgabe	Eine Patientendatei *patients* beinhaltet Körpergröße und -gewicht. Je Patient soll der Body Mass Index berechnet und die zugehörige BMI-Kategorie ermittelt werden (Fuzzy Merge bzgl. Zugehörigkeit zu einem Intervall). Die Kategorien mit ihren oberen Grenzwerten werden in einer zweiten SAS-Datei *bmi_kategorie* bereit gestellt.
Verwendete Features	Mehrere Hash-Objekte Attribute-tag ORDERED Hash-Iterator Methoden FIRST, NEXT, REPLACE und OUTPUT
Lösungsidee	Ein erstes Hash-Objekt *h* enthält die Patientendaten aus *patients* sowie Zusatzattribute für die Ablage des zu berechnenden BMI und der zugehörigen BMI-Kategorien. Ein zweites Hash-Objekt *h_kat* nimmt die Kategorien mit den oberen Intervallgrenzen als Schlüssel auf und wird mit einem Iterator *h_kat_iter* versehen. Je Patient wird beginnend mit dem kleinsten Kategoriewert und in aufsteigender Reihenfolge geprüft, ob der berechnete BMI in diese Kategorie oder in eine der noch nachfolgenden fällt.

Programmbeispiel:

```
DATA _null_;
   IF 0 THEN SET patients bmi_kategorie;
   CALL MISSING(bmi);
   DCL HASH h();
   h.DEFINEKEY("subjid");
   h.DEFINEDATA("subjid", "sex", "age", "bmi", "kategorie");
   h.DEFINEDONE();
   DCL HASH h_kat(DATASET: "bmi_kategorie", ORDERED: "Y");
   h_kat.DEFINEKEY("bis_bmi");
   h_kat.DEFINEDATA(ALL: "Y");
   h_kat.DEFINEDONE();
   DCL HITER h_kat_iter("h_kat"); /* Iterator */
   DO UNTIL(eof);
      SET patients END = eof;
      bmi = round (weight / (height * 0.01) ** 2, 0.1);
      rc = h_kat_iter.FIRST();
      DO UNTIL(h_kat_iter.NEXT());
         rc = h.REPLACE();
         IF bmi < bis_bmi THEN LEAVE;
      END;
   END;
   h.OUTPUT(DATASET: "patients_bmi_cat");
```

```
RUN;

NOTE: There were 8 observations read from the data set
      WORK.BMI_KATEGORIE.
NOTE: The data set WORK.PATIENTS_BMI_CAT has 50000
      observations and 3 ...
NOTE: There were 50000 observations read from the data set
      WORK.PATIENTS.
```

Durch die Definition von *bis_bmi* (der oberen Intervallgrenze jeder Kategorie) als Schlüssel in dem zweiten Hash-Objekt bei gleichzeitigem Setzen von ORDERED: „Y“ und verwenden eines Hash-Iterators kann in aufsteigender Folge geprüft werden, ob ein berechneter BMI in eine betrachtete Kategorie hineinfällt oder nicht. Durch die Methode REPLACE werden die Patienteninformationen um den berechneten BMI und die jeweils geprüfte BMI-Kategorie ergänzt bis die zutreffende Kategorie ermittelt ist.

Die Eingabedateien sind nachfolgend (auszugsweise) dargestellt:

work.patients (obs = 15)

Beob.	subjid	age	sex	height	weight
1	PAT_000001	86	F	168	66.0
2	PAT_000002	59	F	168	85.0
3	PAT_000003	86	F	168	55.0
4	PAT_000004	53	M	187	80.1
5	PAT_000005	87	F	173	62.0
6	PAT_000006	86	M	180	82.8
7	PAT_000007	91	F	150	78.0
8	PAT_000008	86	F	180	50.0
9	PAT_000009	40	M	184	50.9
10	PAT_000010	52	M	170	96.3
11	PAT_000011	32	M	186	76.5
12	PAT_000012	58	F	183	65.0
13	PAT_000013	32	M	173	58.8
14	PAT_000014	30	F	161	89.0
15	PAT_000015	76	F	174	55.0

work.bmi_kategorie

Beob.	Kategorie	bis_bmi
1	0.1 - Starkes Untergewicht (< 16)	16.0
2	0.2 - Mäßiges Untergewicht (16 - < 17)	17.0
3	0.3 - Leichtes Untergewicht (17 - < 18,5)	18.5
4	1 - Normalgewicht (18,5 - < 25)	25.0
5	2 - Übergewicht (25 -< 30)	30.0
6	3.1 - Fettleibigkeit Grad I (30 - < 35)	35.0
7	3.2 - Fettleibigkeit Grad II (35 - < 40)	40.0
8	3.3 - Fettleibigkeit Grad III (>= 40)	999.0

Die ersten 15 Beobachtungen der Ausgabedatei zeigen die um den BMI und die BMI-Kategorie ergänzten Sätze. Erkennbar ist die unsortierte Patientenfolge –die Ausgabe erfolgt in der zufälligen Reihenfolge, wie die Schlüssel im Hash-Objekt abgelegt wurden.

work.patients_bmi_cat (obs = 15)

Beob.	subjid	sex	age	bmi	Kategorie
1	PAT_000188	F	49	13.2	0.1 - Starkes Untergewicht (< 16)
2	PAT_000471	M	53	38.5	3.2 - Fettleibigkeit Grad II (35 - < 40)
3	PAT_000544	F	96	18.9	1 - Normalgewicht (18,5 - < 25)
4	PAT_000617	M	81	33.5	3.1 - Fettleibigkeit Grad I (30 - < 35)
5	PAT_000900	M	21	31.0	3.1 - Fettleibigkeit Grad I (30 - < 35)
6	PAT_001273	F	32	15.6	0.1 - Starkes Untergewicht (< 16)
7	PAT_001346	F	77	30.1	3.1 - Fettleibigkeit Grad I (30 - < 35)
8	PAT_001419	F	81	26.8	2 - Übergewicht (25 -< 30)
9	PAT_001702	F	67	28.1	2 - Übergewicht (25 -< 30)
10	PAT_001999	M	50	23.0	1 - Normalgewicht (18,5 - < 25)
11	PAT_002075	F	54	14.3	0.1 - Starkes Untergewicht (< 16)
12	PAT_002148	M	54	27.5	2 - Übergewicht (25 -< 30)
13	PAT_002431	F	98	33.8	3.1 - Fettleibigkeit Grad I (30 - < 35)
14	PAT_002504	F	40	32.8	3.1 - Fettleibigkeit Grad I (30 - < 35)
15	PAT_003160	F	81	21.8	1 - Normalgewicht (18,5 - < 25)

3.8 Top-<n>%

Aufgabe	Es sollen in Fortsetzung des Beispiels 3.6 diejenigen 5% der Patienten mit dem höchsten BMI ermittelt werden. Die absolute Zahl steht somit vorab nicht fest, sondern ergibt sich erst zur Laufzeit aus der Anzahl der eingelesenen Patienten.
Verwendete Features	Mehrere Hash-Objekte Attribute-tag ORDERED Attribut NUM_ITEMS Hash-Iterator Methoden FIRST, LAST, NEXT, PREV, DELETE und REMOVE
Lösungsidee	Ein zusätzliches drittes Hash-Objekt *h_top* nimmt die im Hash-Objekt *h* vorliegenden Patienteninformationen einschließlich BMI und BMI-Kategorie auf und verwendet den BMI-Wert als Schlüssel in absteigender Reihenfolge. Das Objektattribut NUM_ITEMS beinhaltet die Zahl der Einträge in diesen Hash-Objekten – daraus kann berechnet werden, wie viele Patienten 5% bzw. 95% der Gesamtheit ausmachen.

	Mitte Hilfe des Iterators werden von hinten (mit den kleinsten BMI beginnend) 95% der Patienteneinträge aus dem Hash-Objekt entfernt. Es verbleiben diejenigen 5% der Patienten mit den höchsten BMI.

Programmbeispiel:

```
    ...
    DCL HASH h_top (ORDERED: "D");
    h_top.DEFINEKEY("bmi", "subjid");
    h_top.DEFINEDATA("bmi","subjid","age","sex","kategorie");
    h_top.DEFINEDONE();
    DCL HITER h_iter("h");         /*Iterator über subjid */
    DCL HITER h_top_iter("h_top"); /*Iterator üb. bmi, subjid*/
    anz = h.NUM_ITEMS;             /* Umkopieren */
    h_iter.FIRST();
    DO i = 1 TO anz;
       rc = h_top.ADD();
       rc = h_iter.NEXT();
    END;
    rc = h.DELETE(); rc = h_iter.DELETE();  /*Freigabe*/
    h_top_iter.LAST();
    DO i = 1 TO INT(anz * 0.95);       /*95% weg */
       bmi_del = bmi;
       subjid_del = subjid;
       rc = h_top_iter.PREV();
       rc = h_top.REMOVE(key: bmi_del, key: subjid_del);
    END;
    h_top.OUTPUT(DATASET: "patients_bmi_top5pct");
RUN;

NOTE: The data set WORK.PATIENTS_BMI_TOP5PCT has 2500
      observations ...
```

Das Hash-Objekt-Attribut NUM_ITEMS beinhaltet die Zahl der Einträge eines Hash-Objekts und wird der variablen *anz* zugewiesen.
Unter Verwendung von *anz* werden zunächst alle Patientendaten aus dem von dem Hash-Objekt *h* in das Hash-Objekt *h_top* umkopiert Das ist erforderlich, um eine andere Sortierreihenfolge (absteigend nach BMI) zu realisieren.
Da die Hash-Objekt *h* und der zugehörige Iterator *h_iter* danach nicht mehr benötigt werden, erfolgt deren Löschen unter Anwendung der DELETE-Methode. Der von ihnen belegte Speicherplatz wird dadurch freigegeben.
Anschließend werden – beginnend mit dem letzten Eintrag, also dem kleinsten BMI – 95% aller Einträge aus dem Hash-Objekt *h_top* entfernt. Hierzu wird die REMOVE-Methode verwendet. Da ein von einem Iterator allokierter Eintrag nicht gelöscht werden kann, erfolgt das Löschen um eine Iteration versetzt. Übrig bleiben 5% der ursprünglichen Einträge. Im Beispiel verbleiben aus den ursprünglich 50.000 Patienten 2.500 Patienten in den Top-5%.

work.patients_bmi_top5pct (obs = 15)

Beob.	bmi	subjid	age	sex	Kategorie
1	42.2	PAT_035625	56	M	3.3 - Fettleibigkeit Grad III (>= 40)
2	42.2	PAT_025228	29	M	3.3 - Fettleibigkeit Grad III (>= 40)
3	42.2	PAT_004461	32	M	3.3 - Fettleibigkeit Grad III (>= 40)
4	42.1	PAT_010372	37	M	3.3 - Fettleibigkeit Grad III (>= 40)
5	42.0	PAT_047896	89	M	3.3 - Fettleibigkeit Grad III (>= 40)
6	41.9	PAT_046186	27	M	3.3 - Fettleibigkeit Grad III (>= 40)
7	41.9	PAT_036580	89	M	3.3 - Fettleibigkeit Grad III (>= 40)
8	41.9	PAT_034605	57	M	3.3 - Fettleibigkeit Grad III (>= 40)
9	41.9	PAT_003347	34	M	3.3 - Fettleibigkeit Grad III (>= 40)
10	41.7	PAT_017262	95	M	3.3 - Fettleibigkeit Grad III (>= 40)
11	41.7	PAT_014023	70	M	3.3 - Fettleibigkeit Grad III (>= 40)
12	41.6	PAT_024777	55	M	3.3 - Fettleibigkeit Grad III (>= 40)
13	41.6	PAT_007975	37	M	3.3 - Fettleibigkeit Grad III (>= 40)
14	41.5	PAT_028157	93	M	3.3 - Fettleibigkeit Grad III (>= 40)
15	41.5	PAT_014799	19	M	3.3 - Fettleibigkeit Grad III (>= 40)

3.9 Gruppenstatistiken

Aufgabe	Es sollen in Fortsetzung des Beispiels 3.8 aus den Top-5% je Geschlecht die Zahl der Patienten und deren BMI-Durchschnitt bestimmt werden.
Verwendete Features	Attribute-tag MULTIDATA Methode DO_OVER
Lösungsidee	Ein zusätzliches viertes Hash-Objekt *h_top_sex* nimmt die im hash-Objekt *h_top* vorliegenden Patienteninformationen einschließlich BMI und BMI-Kategorie auf und verwendet das Geschlecht als Schlüssel. Je Geschlecht werden mittels DO_OVER-Methode nacheinander alle dazu vorliegenden Einträge ausgelesen, die Patienten gezählt und die BMI-Werte aufaddiert. Abschließend wird der Durchschnitt ermittelt und erfolgt die Ausgabe der Statistiken in das SAS-Log

Programmbeispiel:

```
    ...
    DCL HASH h_top_sex (MULTIDATA: "Y");
    h_top_sex.DEFINEKEY("sex");
    h_top_sex.DEFINEDATA("sex", "bmi");
    h_top_sex.DEFINEDONE();
    h_top_iter.FIRST();
    DO i = 1 to h_top.NUM_ITEMS;            /* Umkopieren */
       rc = h_top_sex.ADD();
       rc = h_top_iter.NEXT();
    END;
    DO sex = "F", "M";
       anz_subj = 0;
       summe_bmi = 0;
                                  /* Alle mit gleichem KEY */
       DO WHILE (h_top_sex.DO_OVER(KEY: sex) = 0);
          anz_subj = anz_subj + 1;
          summe_bmi = summe_bmi + bmi;
       END;
       mean_bmi = summe_bmi / anz_subj;
       PUT "***** " sex= anz_subj= COMMAX5. @28 mean_bmi= 4.1;
    END;
RUN;

***** sex=F anz_subj=349   mean_bmi=37.8
***** sex=M anz_subj=2.151 mean_bmi=38.4
```

Das Hash-Objekt *h_top_sex* wird mit der MULTIDATA-Eigenschaft deklariert, so dass je Geschlecht Mehrfacheinträge ermöglicht werden. Es muss von *h_top* (Schlüssel ist hier der BMI) auf *h_top_sex* umkopiert werden, um das Geschlecht als Schlüssel verwenden zu können.
Die DO_OVER-Methode stellt eine Schleifenfunktion über alle Einträge mit einem gemeinsamen Schlüsselwert zur Verfügung. In dem Programmbeispiel wird die DO_OVER-Methode in der Form genutzt, dass der Schlüsselwert direkt vorgegeben wird (man kann an dieser Stelle auch explizit einen Wert eintragen oder auch eine andere variable mit passendem Inhalt verwenden).
Ungeachtet dessen, dass in den vorherigen Beispielen die Objektmethoden nur mit impliziter Schlüsselwertdefiniton aufgerufen wurden, können auch dort KEY- und DATA-Werte im Aufruf vorgegeben werden.

4 Fazit

Die Beispielaufgaben sind natürlich auch auf anderem Wegen lösbar - erfordern dafür jedoch ggf. zusätzliche Verarbeitungsschritte (PROC und DATA)

Das Hash-Objekt erweitert die Möglichkeiten des DATA Steps. Es ist insbesondere dann interessant, wenn die unterschiedlichen Techniken in einem DATA Step kombiniert werden sollen.

Das Hash-Objekt ermöglicht eine relativ schlanke Programmierung (Definitionsteil manchmal länger als Logikteil).

Es ist zwischen Key-bezogenen und MULTIDATA-bezogenen Methoden zu unterscheiden (REMOVE löscht z.B. alle Einträge zu einem vorgegebenen Schlüssel, REMOVEDUP nur den aktuellen multiplen Dateneintrag innerhalb eines Schlüssels). Etwas Vorsicht ist walten zu lassen, zumal die Methodennamen nicht immer eine eindeutige Unterscheidung zulassen.

Das Handling ist teilweise ist insbesondere bei MULTIDATA-Anwendungen etwas „sperrig“. Die Möglichkeit einer automatischen Umstrukturierung bei geändertem Schlüssel ist wünschenswert.

Anschauen lohnt sich - nicht nur wegen der Geschwindigkeit!

Literatur

[1] SAS 9.4 Language Reference: Concepts, Fourth Edition.