Lookup und Hash-Objekt – eine Einführung

Michael Fabritius
Basler Versicherungen AG
Aeschengraben 21
CH-4002 Basel
michael.fabritius@baloise.ch

Carlo van de Rijt
Basler Versicherungen AG
Aeschengraben 21
CH-4002 Basel
carlo.van_de_rijt@baloise.ch

Zusammenfassung

Seit dem Release 9.1 gibt es bei SAS stabile Methoden, um Hash-Objekte zu verwenden. Dieses bietet unter anderem neue und schnelle Möglichkeiten, Daten anzureichern. Obwohl die Verwendung des Hash-Objekts in nachfolgenden Releases weiter verbessert wurde, kennen viele SAS-Programmierer diese Möglichkeiten nicht. Entwickler, die mit dem Data-Integration-Studio arbeiten, haben mit der Lookup-Transformation ebenfalls eine Variante zur Verwendung des Hash-Objekts verfügbar.
Anhand von Codebeispielen werden die Grundlagen des Hash-Objekts erklärt und welche Gründe zur Verwendung motivieren. Zielgruppe sind SAS-Programmierer, die dies bisher noch nicht oder unbewusst mittels Lookup-Transformation verwendet haben.
Einfache Anwendungsbeispiele zeigen die vielfältigen Verwendungsmöglichkeiten, auch jenseits von Joins. Neben den Vorteilen werden mögliche Stolperfallen aufgezeigt, die den Einstieg in die Verwendung von Hash-Objekten erschweren können.
Der Vortrag soll einen leichten Einstieg in die Thematik ermöglichen und dazu anregen, bei der Arbeit mit SAS auch das Hash-Objekt in Betracht zu ziehen.

Schlüsselwörter: SAS, SAS Base, DataStep, Hash-Objekt

1 Hintergrund

Obwohl bereits mit SAS 9.1 – also bereits Ende 2003 – eine stabile und einfache Verwendung von Hash-Objekten möglich ist, wird dies bei der reinen SAS-Base-Programmierung recht selten verwendet. Dies beruht einerseits auf Erfahrungswerten im Firmenumfeld und zeigt sich andererseits beispielsweise bei Lösungsvorschlägen in der SAS Community (communities.sas.com). SAS-Entwicklern, die mit dem SAS Data Integration Studio arbeiten, steht dort die Lookup-Transformation bereit, welche ebenfalls Code mit Hash-Objekten erzeugt. Oftmals wird diese Transformation ohne weiteres Hintergrundwissen verwendet.
Dieses Paper soll eine Einführung zum Hash-Objekt in SAS sein und zur weiteren Einarbeitung ermutigen. Es wurde im Rahmen eines Vortrags bei der KSFE- Konferenz 2020 erstellt.

2 Anwendungsfälle

Das SAS Hash-Objekt ist eine Datenstruktur, die eine effiziente Speicherung und Suche von Daten in-memory ermöglicht. Damit lassen sich eine Vielzahl von Problemstellungen performant lösen. Häufige Anwendungen umfassen:

- Datenanreicherung
- Sortierung
- Zählungen
- Iteration über Datensätze
- Aussteuerung bestimmter Datensätze

3 Eigenschaften

Folgende Eigenschaften zeichnen das Hash-Objekt aus:

- Es handelt sich um eine Datenstruktur, die aus mindestens einem Schlüsselelement und optionalen Datenelementen besteht.
- Das Hash-Objekt wird innerhalb eines DataStep erzeugt und existiert nur, während dieser verarbeitet wird.
- Die Verarbeitung erfolgt ausschließlich in-memory. Es muss also ausreichend Arbeitsspeicher für alle Inhalte vorhanden sein. Es findet kein Swapping auf andere Speicherorte (z.B. den Work- oder Utility-Bereich) statt.
- Die Speicherbelegung erfolgt dynamisch. Es ist nicht notwendig, einen bestimmten Bereich vorab zu reservieren.
- In die theoretischen Hintergründe, die bei der Verarbeitung des Hash-Objekts eine Rolle spielen, braucht sich der Entwickler bei der Verwendung in SAS nicht einzuarbeiten. Diese besagen jedoch, dass die Suchdauer auf der Datenstruktur theoretisch unabhängig von der Anzahl der gespeicherten Sätze ist.
- Als Objekt der DATA Step Component Objects verhält sich die Programmierung etwas abweichend zu der gewohnten DataStep-Programmierung. Insbesondere die nachfolgend gezeigte Verwendung von Methoden kann etwas Eingewöhnung erfordern.

3.1 Hashing-Logik

Die Verarbeitung von Daten in einem Hash-Objekt ist wie folgt:

- Eine Hashfunktion ordnet Quellwerte auf Zielwerte. In der Regel ist die Menge der Zielwerte kleiner der Menge der Quellwerte. SAS führt die geeignete Hash-Funktion intern aus. Hier kann und muss ein Entwickler nicht eingreifen.
- Die mittels Hash-Funktion erzeugten Zielwerte dienen dazu, einen Speicherort in sogenannten Buckets zuzuweisen. Somit ergibt sich eine schnelle Suchmöglichkeit, wenn man den Suchwert die gleiche Hash-Funktion durchlaufen lässt.

- Bei den erzeugten Zielwerten kann es ggf. zu Kollisionen kommen, wenn mehrere Werte einem Bucket zugeordnet werden. Dies wird durch das Collision-Handling bearbeitet, welches bei der Programmierung nicht weiter berücksichtigt werden muss.

Die Verarbeitung beim Speichern von Daten in ein Hash-Objekt erfolgt also in den Schritten Hashing und Collision-Handling. Die Struktur eines Hash-Objekts ist bestimmt durch die Anzahl Buckets, in die die Daten einsortiert werden sowie einer balancierten binären Baumstruktur, die in jedem Bucket vorliegt und mögliche Kollisionen abfängt. Die Details gehen jedoch über die Zielsetzung dieses Papers hinaus.

4 Verwendung eines Hash-Objekts

Die typischen Schritte zur Verwendung eines Hash-Objekts sind:

- Deklaration und Instanziierung eines Hash-Objekts.
- Bearbeiten der Daten innerhalb eines Hash-Objekts.
- Suchen innerhalb eines Hash-Objekts.
- Iterieren über ein Hash-Objekt (mittels eines zu definierenden Iterators)
- Vergleich von Hash-Objekten
- Daten von Hash-Objekten ausgeben

Nachfolgend werden diese typischen Schritte anhand einfachster Codebeispiele aufgeführt. Es handelt sich nicht um eine vollständige Dokumentation aller Möglichkeiten von SAS-Hash-Objekten.
Nachfolgender Beispielcode ist immer als in einen DataStep eingebettet zu betrachten.

4.1 Deklaration und Instanziierung

Die Deklaration eines Hash-Objekts erfolgt mittels eines DECLARE-Statements. Dieses kann auch direkt dazu benutzt werden, um eine Instanz im Speicher zu erzeugen. Falls dies nicht direkt erfolgt, muss die Instanz mittels des _NEW_ Operators erzeugt werden. Weitere notwendige Angaben sind die Definition eines Schlüssels, sowie die Beendigung der Definition. Die Definition von Datenfeldern ist optional.
Beispiel für die Verwendung innerhalb eines DataStep:

```
if _n_ = 1 then
  do;
    declare hash h_obj;
    h_obj = _new_ hash();
    rc = h_obj.definekey("abteilung_id");
    rc = h_obj.definedata("abteilung_name");
    rc = h_obj.definedone();
  end;
```

Eine praktische Vorgehensweise ist es, nur beim ersten Durchlauf eines DataStep ein Hash-Objekt zu erzeugen, was hier in der ersten Zeile sichergestellt ist.

Im Beispiel wird ein Hash-Objekt mit dem Bezeichner "h_obj" erzeugt. Dieses wird anschließend durch den _NEW_ Operator instanziiert. Danach folgen die Methodenaufrufe des erzeugten "h_obj", die jeweils einen Rückgabewert bei erfolgreicher Ausführung liefern. Dieser Rückgabewert wird im Beispiel in der Variablen "rc" (für ReturnCode) gespeichert, hier jedoch nicht weiterverwendet. Zunächst wird ein Schlüsselfeld "abteilung_id" definiert, anschließend ein Datenfeld "abteilung_name". Abschließend wird die Definition des Hash-Objekts abgeschlossen.
Zu beachten gilt, dass die Felder ggf. schon im Vorfeld im DataStep definiert sein sollten, um Länge und Format festzulegen.

Der alternative Aufruf zur Deklaration und gleichzeitigem Instanziieren wäre für dieses Beispiel wie folgt:

```
declare hash h_obj();
```

In der hier leeren Klammer können weitere Codeelemente für das Hash-Objekt aufgeführt werden. So lässt sich beispielsweise ein DataSet angeben, welches direkt in das Hash-Objekt geladen werden soll. Weiterhin kann man beispielsweise definieren, ob ein Schlüssel ein- oder mehrfach vorkommen darf oder ob die Einträge sortiert werden sollen.
Nachfolgend ein Beispiel zum Laden des DataSet "work.abteilung", bei dem mehrere Datensätze mit dem gleichen Schlüsselfeld in das Hash-Objekt übernommen werden:

```
declare hash h_obj(dataset: "work.abteilung", multidata: "Y");
```

4.2 Bearbeiten der Daten

Daten können wie oben beschrieben bereits bei der Definition eines Hash-Objekts aus einem DataSet geladen werden. Daneben besteht die Möglichkeit weitere Daten an beliebiger Stelle im DataStep hinzuzufügen, Daten zu ersetzen oder zu löschen. Hierfür gibt es die Methoden

- *add()* – fügt neue Daten zum Hash-Objekt hinzu
- *ref()* – fügt neue Daten zum Hash-Objekt hinzu, falls der Schlüssel noch nicht im Hash-Objekt vorhanden ist
- *replace()* – ersetzt Daten zu einem gegebenen Schlüssel
- *remove()* – entfernt den Eintrag zu einem gegebenen Schlüssel
- *clear()* – entfernt alle Einträge aus dem Hash-Objekt. Das Hash-Objekt selbst bleibt erhalten.

In unserem Beispiel möchten wir einen weiteren Datensatz mit abteilung_id "12" und abteilung_name "Marketing" zu unserem Hash-Objekt hinzufügen. Dazu gibt es verschiedene Möglichkeiten, zwei Varianten werden näher beschrieben:

- Befüllen der Variablen, wie sie auch im Hash-Objekt existieren und anschließendes ADD()

```
abteilung_id = 12;
abteilung_name = "Marketing";
rc = h_obj.add();
```

  In diesem Fall nutzt die Methode ADD() die im PDV vorliegenden relevanten Variablen und fügt die Daten dem Hash-Objekt hinzu. Weitere im PDV vorliegende Variablen, die nicht im Hash-Objekt definiert sind, werden nicht berücksichtigt.

- Übergabe der Werte (direkt oder aus bereits vorliegenden Variablen) in der Methode ADD()

```
rc = h_obj.add(key: 12, data: "Marketing");
```

4.3 Suchen innerhalb eines Hash-Objekts

Für das Suchen von Werten innerhalb eines Hash-Objekts wird die Methode FIND() verwendet. Dabei können analog zum vorherigen Beispiel mit ADD() Schlüsselfelder mitgegeben werden oder nicht. Die gefundenen Datenfelder werden aus dem Hash-Objekt bei erfolgreicher Suche in die Variablen des PDV übernommen. Dies kann dazu verwendet werden, um Werte anzureichern oder anhand des ReturnCodes zu prüfen, ob ein Schlüssel im Hash-Objekt vorhanden ist.
Beispiel:

```
rc = h_obj.find(key: 12);
```

Wenn der Schlüssel (hier 12) gefunden wird, so erhält die Variable "rc" den Wert 0 und alle Data-Variablen des Hash-Objekt werden entsprechend in den PDV befüllt.
Wird der Schlüssel nicht gefunden, so erhält die Variable "rc" den Wert 160038. In der Praxis bewährt es sich, darauf zu prüfen, ob der ReturnCode gleich 0 (erfolgreiche Ausführung der Methode) oder ungleich 0 (nicht erfolgreiche Ausführung) ist, da die Methoden des Hash-Objekts verschiedene Werte zurückliefern, die seitens SAS nicht offiziell dokumentiert sind.
Anstatt den Schlüssel explizit zu übergeben, kann dies auch mittels einer Variablen erfolgen oder die Schlüsselvariable des Hash-Objekts wird zuvor auf den zu suchenden Schlüssel gesetzt. Im letzten Fall ist keine Angabe in Klammern der Methode erforderlich:

```
abteilung_id = 12;
rc = h_obj.find();
```

4.4 Iterieren über ein Hash-Objekt

Je nach Aufgabenstellung kann es praktisch sein, sich zeilenweise im Hash-Objekt zu bewegen. Dafür gibt es das sogenannte Iterator-Objekt, welches zu einem Hash-Objekt definiert werden muss. Dazu kommt wieder das DECLARE-Statement zum Einsatz und erneut gibt es die Möglichkeit der separaten Instanziierung mittels "_NEW_" oder der Instanziierung direkt mit dem DECLARE-Statement. Anstatt eines "declare hash" wird nun ein Hash-Iterator, kurz "hiter", definiert:

```
declare hiter h_iter("h_obj");
```

Anzumerken ist, dass das Iterator-Objekt auf ein bestimmtes Hash-Objekt zeigt und dieses daher mit angegeben werden muss. Im Beispiel wurde ein Hash-Iterator "h_iter" erzeugt, der auf das Hash-Objekt "h_obj" zeigt.

Der Hash-Iterator bietet die Methoden:

- *first()* – kopiert die Datenfelder des ersten Eintrags im Hash-Objekt in den PDV
- *next()* – kopiert die Datenfelder des nächsten Eintrags im Hash-Objekt in den PDV
- *prev()* – kopiert die Datenfelder des vorherigen Eintrags im Hash-Objekt in den PDV
- *last()* – kopiert die Datenfelder des letzten Eintrags im Hash-Objekt in den PDV
- *setcur()* – setzt den Iterator auf einen speziellen Schlüssel. Dieser muss beim Aufruf mitgegeben werden.
- *delete()* – löscht das Iterator-Objekt

Beispiel:

```
do while (h_iter.next() = 0);
  put _all_;
end;
```

Der aufgeführte Loop ruft die NEXT-Methode des Iterators auf. Da die FIRST-Methode nicht aufgerufen wird, setzt der Iterator beim Aufruf von NEXT auf dem ersten Schlüssel auf und kopiert die Daten aus dem Hash-Objekt in den PDV. Bei den nächsten Aufrufen werden entsprechend die Daten der nachfolgenden Einträge im Hash-Objekt geladen. Ist der Aufruf erfolgreich, so wird als Rückgabewert 0 geliefert. Die Bedingung des Loops ist somit erfüllt, so lange es noch Elemente im Hash-Objekt gibt.
In diesem einfachen Beispiel wird innerhalb des Loops lediglich der Inhalt des PDV ausgegeben.

4.5 Vergleich von Hash-Objekten

Mit einem einfachen Statement können Hash-Objekte verglichen werden. Dadurch lässt sich beispielsweise ein Compare innerhalb eines DataStep abbilden.

```
rc = h_obj1.equals(hash: 'h_obj2', result: eq);
```

In diesem Fall werden zwei Hash-Objekte verglichen "h_obj1" und "h_obj2". Das Ergebnis wird in die Variable "eq" geschrieben. Diese erhält den Wert 1, wenn die Hash-Objekte gleich sind, andernfalls ist der Wert 0. Gleichheit bedeutet in diesem Zusammenhang:

- HASHEXP beider Hash-Objekte ist gleich
- die Anzahl der Einträge (num_items) ist identisch
- Key und Data sind jeweils gleich definiert
- alle Einträge sind in beiden Hash-Objekten mit gleichen Werten und in gleicher Reihenfolge vorhanden

4.6 Daten von Hash-Objekten ausgeben

Die Ausgabe von Hash-Objekten erfolgt mittels der OUTPUT-Methode. Diese wird für ein Hash-Objekt aufgerufen und braucht als Option den Namen des Datasets, das geschrieben werden soll:

```
rc = h_obj.output(dataset: "work.output");
```

Die Ausgabe erfolgt dabei ggf. nach der im Hash-Objekt angegebenen Sortierung (auf- oder absteigend). Die Schlüssel des Hash-Objekts werden nicht ausgegeben, daher sind diese bei Bedarf als Schlüssel und als Datenelement zu definieren.
Dieses einfache Statement ist ein praktisches Werkzeug, denn es können auch mehrere Datasets mit verschiedenen Dataset-Optionen in einem Statement erzeugt werden:

```
rc = h_obj.output(
           dataset: "work.output1 (where=(abteilung_id = 1))",
           dataset: "work.output2 (where=(abteilung_id = 2))"
           );
```

Neben der WHERE-Option können RENAME, DROP und KEEP ausgeführt werden. Weiterhin lässt sich die Ausgabe dynamisch aufbauen:

```
do loop = 1 to 100;
 rc = h_obj.output(dataset:
 cats("work.output_",loop,"(where=(abteilung_id=",loop,"))"));
end;
```

Obiger Code würde die Datasets work.output_1 bis work.output_100 erzeugen und über die aufgebaute WHERE-Option jeweils nur die Datensätze mit entsprechender abteilung_id in das jeweilige Dataset ausgeben. Dazu muss, wie oben beschrieben, die Variable abteilung_id als Datenelement im Hash-Objekt definiert sein.

5 Performance-Betrachtungen

Zur Überprüfung der theoretischen Grundlage wurden einige Beispielbetrachtungen hinsichtlich der Performance durchgeführt.
Dazu muss angemerkt werden, dass allgemeingültige Aussagen zur Performance schwierig zu treffen sind, da diese von einer Vielzahl individueller Faktoren abhängig ist. Beispielsweise den Systemoptionen, der Hardware und der sonstigen Last auf dem System.
Die Dauer der nachfolgenden Beispiele wurde ermittelt, indem die Systemzeit vor der Ausführung eines entsprechenden DataStep mit der Systemzeit nach der Ausführung abgeglichen wurde. Um die aktuelle Systemlast weniger stark zu berücksichtigen, wurden jeweils 10 Durchläufe durchgeführt und der Mittelwert der Ergebnisse dargestellt.

5.1 Performance bei der Erzeugung

Zunächst wird die Dauer bei der Erstellung von Hash-Objekten betrachtet. Dies wird zur Veranschaulichung mit der jeweiligen Größe des Hash-Objekts abgebildet. Hierzu wird ein Hash-Objekt (1) bestehend aus einem numerischen Schlüssel Länge 8 und einem alphanumerischen Datenfeld Länge 50 erzeugt. Zum Vergleich wird ein zweites Hash-Objekt (2) mit 2 numerischen Schlüsseln Länge 8 und 5 alphanumerischen Datenfeldern Länge 50 erzeugt. Für eine zunehmende Anzahl Sätze jeweils die Dauer des DataStep und die Größe des Hash-Objekts ermittelt. Daraus ergibt sich folgende Darstellung:

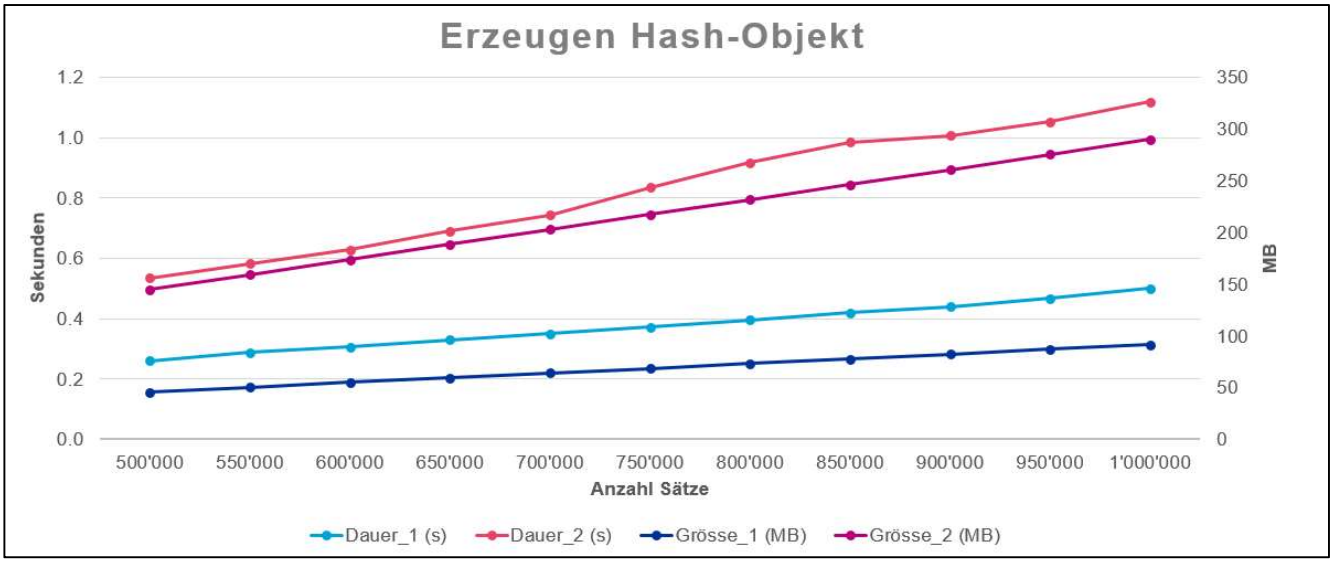

Abbildung 1: Größe und Dauer der Erzeugung zweier Hash-Objekte bei zunehmender Anzahl Sätze

Die Größe wird aus zwei Attributen des Hash-Objekts ermittelt. Die Anzahl der Sätze (`anzahl = h_obj.num_items`) wird mit der Größe eines jeweiligen Satzes im Hash-Objekt (`bytesize = h_obj.item_size`) multipliziert. Die "Item_Size" richtet sich dabei nur nach der Größe eines Eintrags und berücksichtigt nicht den ggf. existierenden

Overhead für die Verwaltung des Hash-Objekts im Speicher. Nicht verwunderlich ist daher der exakt lineare Anstieg der Größe mit zunehmender Anzahl Sätze.
Gut zu sehen ist jedoch, dass die Dauer beim Erzeugen eines Hash-Objekts im Beispiel ebenfalls linear ansteigt und somit eine gute Skalierbarkeit gegeben ist.

5.2 Performance beim Lesen

Zur Prüfung der Performance bei Abfragen auf das Hash-Objekt werden die oben beschriebenen Hash-Objekte (1. ein Schlüssel, ein Datenfeld und 2. zwei Schlüssel, fünf Datenfelder) mit jeweils 1 Mio. Sätzen in einem DataStep zunächst erzeugt, anschließend werden Abfragen auf zufällige Schlüssel durchgeführt.

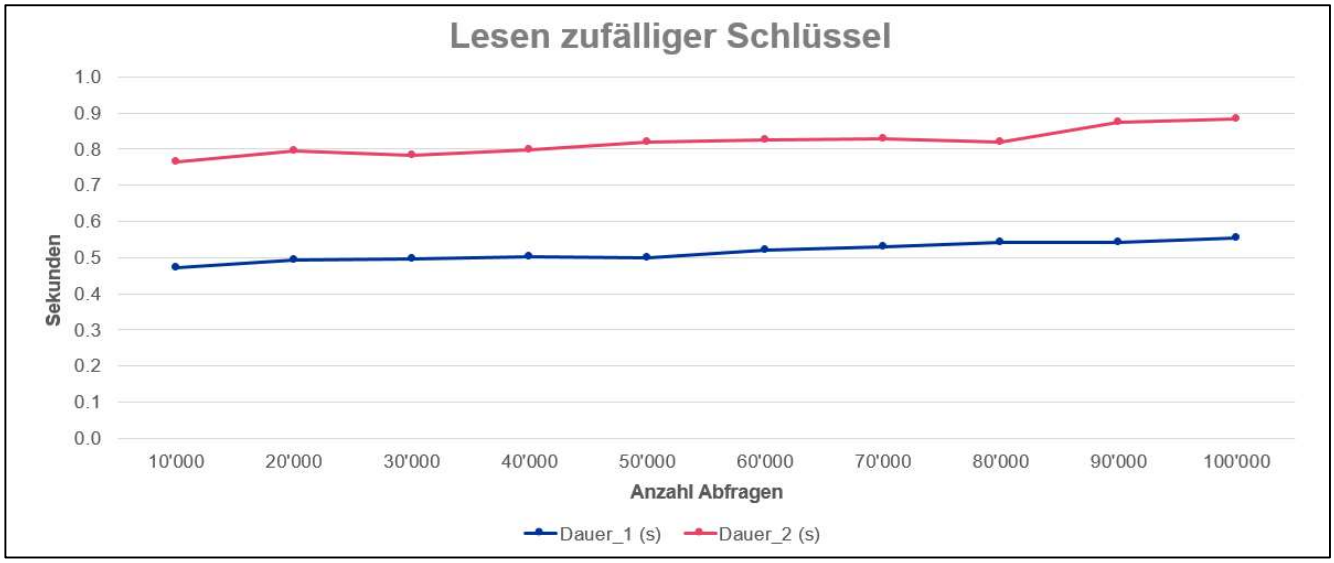

Abbildung 2: Veränderung der Abfragedauer mit steigender Anzahl Abfragen auf zwei Hash Objekte

Hier zeigt sich die Geschwindigkeit, mit der Abfragen auf Hash-Objekte durchgeführt werden. Die Dauer steigt mit zunehmender Anzahl Abfragen kaum an. Der leichte Anstieg bei mehr Abfragen, sowie die unterschiedliche Dauer bei beiden Hash-Objekten lassen sich durch die größere Datenmenge erklären, die jeweils vom Hash-Objekt in die entsprechenden PDV-Variablen kopiert wird.

5.3 Auswirkung von Hashexp

Einfluss auf die Struktur von Hash-Objekten kann man neben der Angabe von Schlüssel- und Datenfeldern über den Parameter "Hashexp" nehmen:

```
declare hash h_obj(hashexp: 5);
```

Der Parameter wirkt sich auf die Anzahl der Buckets aus, die innerhalb des Hash-Objekts angelegt werden (siehe dazu auch den Punkt "Hashing-Logik"). Dabei wird der angegebene Wert als Exponent für 2 genommen, somit ergibt sich im Beispiel mit Hashexp 5 die Berechnung 2 hoch 5 gleich 32. Also stehen in diesem Beispiel 32 Buckets zur Verfügung.

Es können die ganzzahligen Werte von 0 bis 20 als Hashexp verwendet werden. Falls kein Wert angegeben wird, verwendet SAS als Default-Wert 8. Im Data Integration Studio wird der Wert für Hashexp anhand der Anzahl Sätze ermittelt, die in das Hash-Objekt geladen werden sollen. Es wird der 2er Logarithmus der Anzahl Sätze ermittelt und vom Ergebnis der nächsthöhere Integer-Wert genommen: `ceil(log2(nlobs))`

Nachfolgend wurde die Auswirkung von Hashexp auf die Performance betrachtet. Dazu wurde das schon zuvor beschriebene Hash-Objekt mit 2 Schlüsseln und 5 Datenfeldern mit allen möglichen Hashexp-Werten erzeugt. In jeweils 10 Durchläufen wurde die Dauer ermittelt und anschließend der Durchschnitt darüber gebildet. Dies wurde mit 500.000 Sätzen und mit 1 Mio. Sätzen durchgeführt:

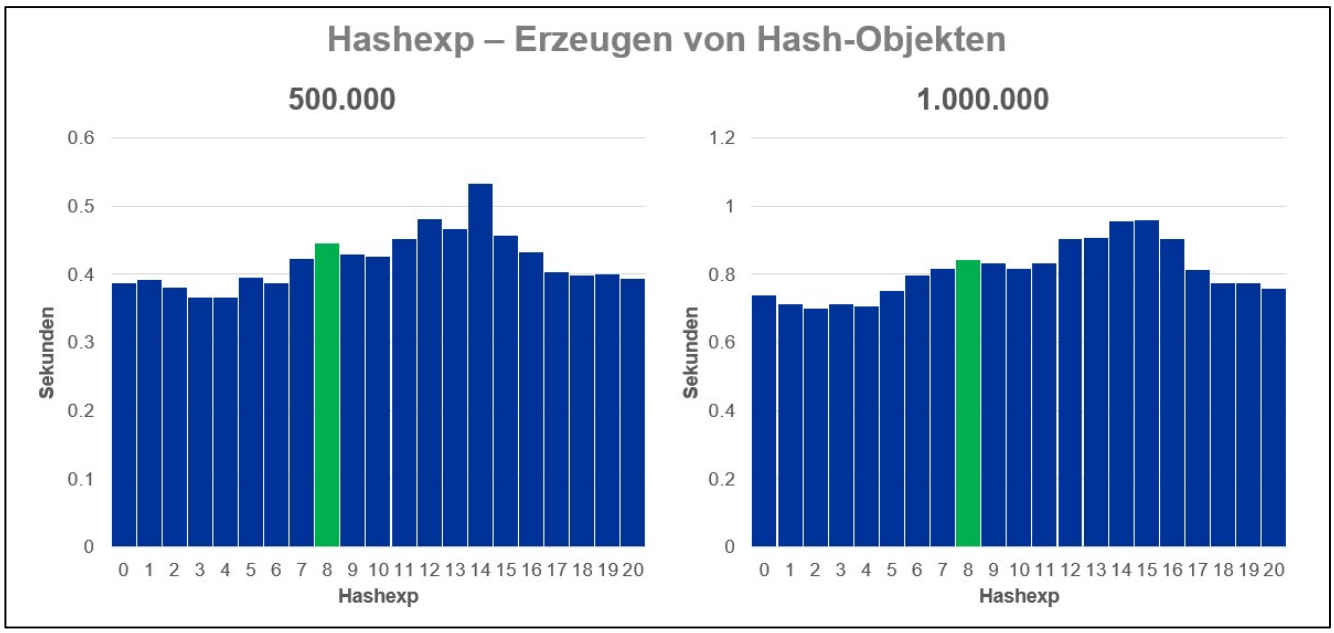

Abbildung 3: Auswirkung von Hashexp auf die Dauer der Erzeugung zweier Hash-Objekte

Es lassen sich – ohne auf die exakten Laufzeiten einzugehen – folgende Schlüsse in Bezug auf den Hashexp-Wert ziehen:

- der Default-Wert 8 (grün eingefärbt) ist nicht immer die performanteste Option
- ein größerer Wert führt nicht immer zu besserer Performance
- kleinere Werte führen nicht immer zu schlechterer Performance – auch nicht bei steigender Datenmenge
- selbst mit nur wenigen Buckets funktioniert das Collision-Handling innerhalb der Buckets sehr schnell

Der Vollständigkeit halber hier noch die Auswertung für das Lesen des kleinen und größeren Hash-Objekts mit jeweils 1 Mio. Einträgen. Diese wurden erstellt und anschließend in jeweils 10 Durchläufe mit 10.000 bis 100.000 Abfragen (in Schritten von 10.000) gemacht und für die Dauer der Durchschnitt der jeweiligen Durchläufe genommen:

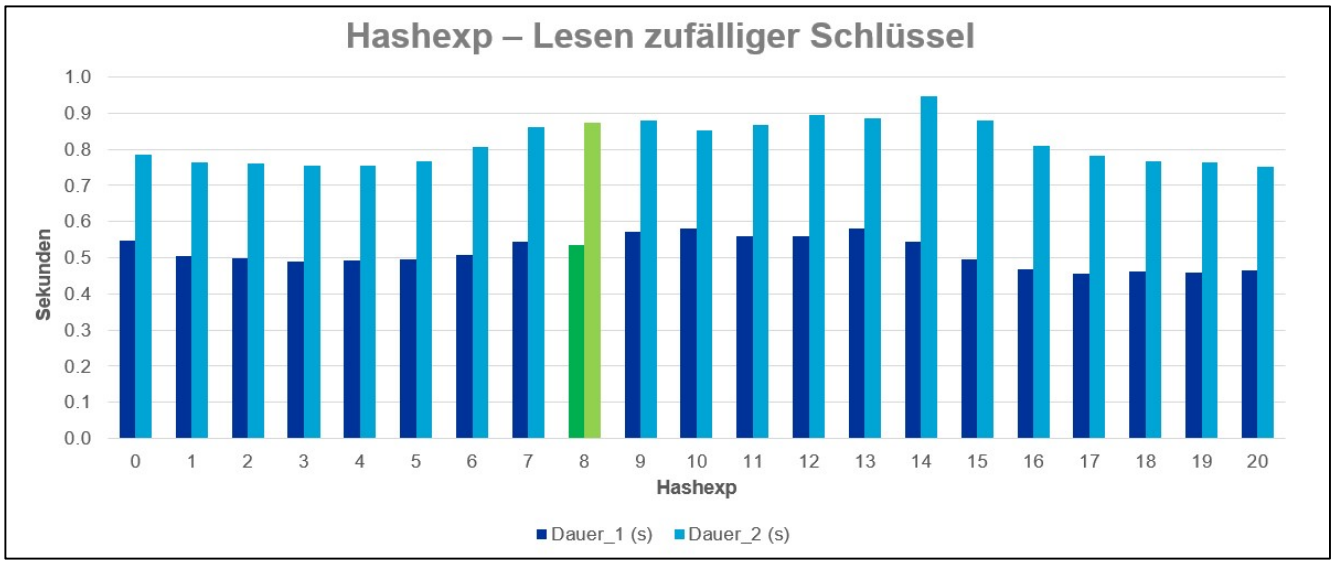

Abbildung 4: Auswirkung von Hashexp auf die Dauer der Abfragen zweier Hash-Objekte

Es zeigt sich, dass auch beim Lesen die gleichen Rückschlüsse gelten, die schon beim Erzeugen des Hash-Objekts aufgeführt wurden.

Natürlich muss hier im Einzelfall genauer geprüft werden, ob die Laufzeit größere Unterschiede aufweist und ob diese für die Verarbeitung relevant sind. Je nachdem, wie oft ein Hash-Objekt in der geplanten Verarbeitung angelegt wird und welche Daten enthalten sein sollen, können die Ergebnisse unterschiedliche Bedeutung erlangen. Bei ersten Programmierversuchen mit dem Hash-Objekt kann Hashexp sicherlich vernachlässigt werden. Mit zunehmender Erfahrung kann sich eine näher Betrachtung – je nach Anwendungsfall – lohnen.

6 Zusammenfassung

Hash-Objekte eignen sich prima für diverse Prozesse von Sortierungen, Datenanreicherung, erweiterte Logik bei Datenanreicherung z.B. mit Fehlerhandling bis hin zu erweiterten Möglichkeiten bei der Ausgabe von Tabellen. Mit wenigen Codebausteinen lassen sich viele Anwendungsfälle abdecken. Insbesondere durch die Berücksichtigung von Rückgabewerten (ReturnCodes) der Hash-Objekt-Methoden können im DataStep praktische bedingte Verarbeitungen ausgeführt werden. Als sinnvolle Ergänzung kann man sich mit dem Iterator-Objekt nochmals gezielt innerhalb der Daten eines Hash-Objekts bewegen.
Dabei wird die komplexe Logik von Hash-Objekten stets durch SAS im Hintergrund erledigt und bleibt dem Anwender somit verborgen.
Die Speicherung und insbesondere das Auslesen der Daten funktionierten sehr schnell.
Das größte Risiko bei der Verwendung stellt der Arbeitsspeicher dar, der für die Daten im Hash-Objekt ausreichen muss.

Anhang: Quellen/weitere Empfehlungen

Quellen für dieses Paper:

SAS® 9.4 Language Reference: Concepts, Sixth Edition
(https://documentation.sas.com/?docsetId=lrcon&docsetTarget=titlepage.htm&docsetVersion=9.4&locale=en)
SAS® 9.4 Component Objects: Reference, Third Edition
(https://documentation.sas.com/?docsetId=lecompobjref&docsetTarget=titlepage.htm&docsetVersion=9.4&locale=en)

Als schnelle Übersicht für den Einstieg gibt es von SAS ein empfehlenswertes Dokument:

Hash Object Tip Sheet
(https://support.sas.com/rnd/base/datastep/dot/hash-tip-sheet.pdf)