SAS DataStep Component Interface – Neue Objekte im DataStep

Sebastian Reimann
viadee Unternehmensberatung GmbH
Anton-Bruchausen-Str. 8
48147 Münster
sebastian.reimann@viadee.de

Zusammenfassung

Über das Component Objects Interface erhält der klassische DataStep neue Funktionalität. Zum einen werden die neuen Möglichkeiten des Hash Objektes näher untersucht. Hier wird gezeigt, wie ab SAS 9.2 mit Duplikaten im Hash Schlüssel umgegangen werden kann und welche Vorteile sich daraus ergeben. Weiterhin wird gezeigt, wie mit Hilfe eines Hash Objektes innerhalb des DataSteps Aggregationen in unsortierten Datasets durchgeführt werden können und die Ergebnisse anschließend sortiert in einem Dataset gespeichert werden. Große Performance-Verbesserungen lassen sich durch die Kombination mehrerer Hash Objekte erreichen.
Nachfolgend werden die Möglichkeiten untersucht, die sich aus dem DataStep Java Interface ergeben. Es wird aufgezeigt, wie mit einfachen Mitteln aus einem SAS DataStep auf eine relationale Datenbank zugegriffen werden kann, ohne hierfür separate SAS Access-Module zu nutzen. Gerade für kleine oder unbekannte Datenbanken kann das Java Interface eine preiswerte Alternative sein.

Schlüsselwörter: Component Interface, DataStep, Hash Objekt, Java Interface, Datenbankzugriff, SAS Access-Modul

1 Optimierter Join durch Hash Objekte

Der DataStep zählt zu den universellsten und vielseitigsten Elementen im Sprachumfang der SAS Sprache. Seit Beginn von SAS bildet er das Fundament vieler ETL Prozesse. Mit ihm ist es möglich, große Datenmengen in kurzer Zeit effektiv zu verarbeiten. Grund hierfür ist die sequentielle Verarbeitung des Eingabedatenstroms und die programmgesteuerte Ausgabe in beliebige Ausgabedatenströme.

Auch die Verknüpfung (Merge) mehrerer Eingabedatenströme zu einem gemeinsamen Ausgabedatenstrom anhand von Schlüsselvariablen zählt zu den Stärken des DataStep. Einzige Voraussetzung ist, dass alle Eingabedaten nach den Schlüsselvariablen sortiert sind oder dass entsprechende Indizes existieren.

Doch was passiert, wenn die Verknüpfung nicht anhand einer einzelnen Schlüsselvariablen vorgenommen werden kann? Beispiel hierfür wäre das Nachlesen des künstlichen

Dimensionsschlüssels für einen Faktendatensatz anhand des fachlichen Schlüssels und des Gültigkeitszeitraums.

1.1 Klassischer Ansatz

Der klassische Ansatz wäre an dieser Stelle die Nutzung eines Proc Sql mit entsprechender Join-Bedingung:

```
01 PROC SQL NOPRINT;
02   CREATE TABLE ERGEBNIS AS
03     SELECT DIM.SURROGATE_KEY
04           ,FAKT.FACH_KEY
05           ,FAKT.DATUM
06           ,FAKT.WERT
07     FROM FAKTEN AS FAKT
08     LEFT JOIN DIMENSION AS DIM
09     ON FAKT.FACH_KEY = DIM.FACH_KEY
10     AND FAKT.DATUM BETWEEN DIM.GUELTIG_VON AND DIM.GUELTIG_BIS;
11 QUIT;
```

Über den Left Join wird sichergestellt, dass alle Fakten-Datensätze in die Ausgabetabelle übernommen werden, unabhängig davon ob ein künstlicher Schlüssel ermittelt werden konnte oder nicht.

Ist es im Anschluss noch notwendig, die Ergebnistabelle in zwei Tabellen (Datensätze mit gültigem künstlichen Schlüssel und Datensätze, für die ein neuer Dimensionseintrag generiert werden muss) aufzuspalten, ist ein weiterer Schritt, zum Beispiel ein DataStep mit zwei Ausgabe Datasets notwendig. Gerade bei sehr großen Datenmengen kann dieser zusätzliche Durchlauf durch die Daten zu hohen I/O-Kosten führen, da die Daten mehrfach geschrieben und gelesen werden.

```
01 DATA ERGEBNIS_MATCH
02      ERGEBNIS_MISS;
03   SET ERGEBNIS;
04   IF SURROGATE_KEY EQ . THEN OUTPUT ERGEBNIS_MISS;
05   ELSE OUTPUT ERGEBNIS_MATCH;
06 RUN;
```

Als letzter Schritt im ETL-Prozess ist es häufig gewünscht, eine Statistik über die verarbeiteten Datensätze zu erzeugen. Diese Statistiken werden häufig zur Beurteilung der Datenqualität und der Verarbeitungsperformance genutzt. Hierbei werden die neu verarbeiteten Datensätze gruppiert und gezählt. Auch hierfür ist ein weiterer Proc Sql notwendig, was wiederum zum Lesen der gesamten Daten führt und somit erneut I/O-Kosten produziert.

```
01 PROC SQL NOPRINT;
02   CREATE TABLE STATISTIK AS
03     SELECT SURROGATE_KEY
04           ,COUNT(*) AS ANZAHL
05     FROM ERGEBNIS
06     GROUP BY SURROGATE_KEY;
07 QUIT;
```

1.2 Hash Objekt Ansatz

Mit Hilfe der Hash Objekte und der Möglichkeit, seit SAS 9.2 Duplikate in Schlüsseln zu verarbeiten, sind sämtliche zuvor genannte Schritte in einem einzelnen DataStep möglich. Dies hat den Vorteil, dass die gesamten Daten nur ein einziges Mal durchlaufen werden müssen und sofort sämtliche Ergebnistabellen erzeugt werden können.

```
01 DATA ERGEBNIS_MATCH(KEEP=SURROGATE_KEY FACH_KEY DATUM WERT)
02      ERGEBNIS_MISS(KEEP=SURROGATE_KEY FACH_KEY DATUM WERT);
03   SET FAKTEN END=EOF;
04   ATTRIB ANZAHL        FORMAT=COMMAX20.
05          SURROGATE_KEY FORMAT=8.;
06   IF _N_ = 1 THEN DO;
07     DECLARE HASH L(DATASET: "DIMENSION", MULTIDATA: "Y");
08     L.DEFINEKEY("FACH_KEY");
09     L.DEFINEDATA("SURROGATE_KEY", "GUELTIG_VON", "GUELTIG_BIS");
10     L.DEFINEDONE();
11     DECLARE HASH S(ORDERED: "A");
12     S.DEFINEKEY("SURROGATE_KEY");
13     S.DEFINEDATA("SURROGATE_KEY", "ANZAHL");
14     S.DEFINEDONE();
15   END;
16   RC = L.FIND();
17   DO WHILE (RC=0);
18     IF GUELTIG_VON <= DATUM <= GUELTIG_BIS THEN LEAVE;
19     SURROGATE_KEY = .;
20     RC = L.FIND_NEXT();
21   END;
22   IF RC EQ 0 THEN ERGEBNIS_MATCH;
23   ELSE OUTPUT ERGEBNIS_MISS;
24   IF S.FIND() NE 0 THEN ANZAHL = 0;
25   ANZAHL + 1;
26   S.REPLACE();
27   IF EOF THEN S.OUTPUT(DATASET: "STATISTIK");
28 RUN;
```

Zu Beginn wird der DataSep mit den zwei Ausgabetabellen definiert. Für jede der beiden Ausgabetabellen kann definiert werden, welche Variablen übernommen werden sollen. Über das Set Statement wird die Faktentabelle eingelesen. Hierbei wird jeder Satz durchlaufen.

Zu Beginn des ersten Durchlaufs werden zwei Hash Objekte definiert. Unter dem Objektnamen `L` wird die Dimensionstabelle als Lookup Tabelle in den Speicher geladen. Über den Zusatz `Multidata: „Y“` wird definiert, dass bei diesem Hash Objekt Duplikate im Schlüssel erlaubt sind und dies nicht zu einem Abbruch des DataSteps beim Laden der Daten führen soll.

Für das Hash Objekt müssen über die Methoden DefineKey und DefineData die Schlüssel- und Datenvariablen definiert werden, anhand derer Werte im Hash Objekt nachgeschlagen bzw. aus dem Hash Objekt ausgelesen werden sollen. In der Lookup Tabelle wird über den fachlichen Schlüssel gesucht. Als Datenvariablen werden sowohl der gesuchte künstliche Schlüssel, als auch die beiden Gültigkeitsvariablen benötigt, da diese später für jeden Satz geprüft werden müssen.

Für die Erzeugung der Statistik-Tabelle wird das zweite Hash Objekt definiert. Über den Zusatz `Ordered: „A“` wird festgelegt, dass bei der späteren Ausgabe des Objektes in ein Dataset die Daten in aufsteigender Reihenfolge, sortiert nach der Schüsselvariable, ausgegeben werden. Auch hier werden über DefineKey und DefineData die Schlüssel- und Datenvariablen definiert. Wichtig ist, dass alle Variablen, die später im Ausgabe-Dataset enthalten sein sollen, auch als Datenelement definiert sind. Aus diesem Grund ist der künstliche Schlüssel sowohl als Schlüssel- als auch als Datenvariable angegeben.

Über den Aufruf `L.FIND()` wird für jeden aus der Eingabetabelle gelesenen Datensatz der künstliche Schlüssel im Hash Objekt nachgelesen. Das Ergebnis des Nachleseprozesses wird in der RC-Variablen gespeichert. Beim Aufruf der FIND-Methode ist es nicht möglich, die Variable für den Hash-Lookup explizit anzugeben. Es ist Voraussetzung, dass die Schlüsselvariable im Hash Objekt mit dem Variablennamen im Eingabe-Dataset namentlich übereinstimmt. Nur so kann der Lookup erfolgen. Ist diese Namensgleichheit nicht gegeben, müssen entsprechende Rename Statements vorgesehen werden, die die Variablennamen vereinheitlichen.

Die FIND-Methode liefert eine 0 als Rückgabewert, wenn der entsprechende Wert im Hash Objekt gefunden wurde. Alle anderen Rückgabewerte bedeuten eine erfolglose Suche. Verbunden mit der erfolgreichen Suche wurden automatisch die Werte der Datenvariablen `SURROGATE_KEY`, `GUELTIG_VON` und `GUELTIG_BIS` in den PDV des aktuell verarbeiteten Datensatzes kopiert. Somit ist es direkt möglich, den Gültigkeitszeitraum des künstlichen Schlüssels anhand der im Datensatz enthaltenen Datums-Variablen zu prüfen.

Liegt das Datum des Datensatzes nicht zwischen den gefundenen Gültigkeitswerten, wird über den `FIND_NEXT()` Methode der nächste Datensatz zu der Schlüssselvariablen gesucht und der Prüfvorgang wiederholt. Wird kein weiterer Satz zu dem Schlüssel gefunden, wird aufgrund des Return Code <> 0 der Schleifendurchlauf abgebrochen.

Die Aufteilung in zwei Ausgabe-Datasets kann leicht anhand des Suchergebnisses in der RC-Variablen erfolgen. Hat diese den Wert 0, so wurde ein korrekter, gültiger künstlicher Schlüssel gefunden, andernfalls nicht.

Anschließend wird für jeden Datensatz über das Statistik- Hash Objekt eine Suche anhand des künstlichen Schlüssels vorgenommen. Wurde ein Datensatz im Hash Objekt gefunden, wird der dort gespeicherte Anzahl-Wert um 1 erhöht und im Hash Objekt ersetzt. Wurde kein Wert für den künstlichen Schlüssel im Hash Objekt gefunden, wird der Anzahl-Wert mit 0 initialisiert und dem Hash Objekt hinzugefügt.

Wurde die Eingabedatenquelle komplett verarbeitet (DataSet Variable EOF ist mit 1 belegt), so wird das Statistik-Hash Objekt in ein SAS Dataset ausgegeben und somit dauerhaft gespeichert.

1.3 Vergleich beider Ansätze

Die Ergebnisse beider Herangehensweisen sind exakt identisch. Bei der ersten Variante unter Kombination des Proc Sql mit dem DataStep wurde lediglich eine zusätzliche, temporäre Tabelle erzeugt, die jedoch nach Abschluss der Verarbeitung problemlos gelöscht werden kann.

Je nach verarbeiteter Datenmenge kann jedoch der Ressourcenverbrauch beider Alternativen deutlich unterschiedlich sein. Gerade bei sehr großen Faktentabellen und überschaubaren Dimensionstabellen stellt die zweite Variante eine performante Alternative dar.

Beide Varianten wurden mit einem identischen, generierten Testdatenbestand auf die Verarbeitungsgeschwindigkeit hin untersucht. Die Faktentabelle enthielt dabei 73,1 Mio Datensätze, die Dimensionstabelle 71.928 Dimensionseinträge wobei das Verhältnis zwischen erfolgreichem Dimensions-Lookup und erfolglosem Lookup in etwa gleich verteilt war.

Tabelle 1: Performancevergleich Variante 1 (Proc Sql / DataStep) zu Variante 2 (Hash Objekte)

Variante 1			**Variante 2**		
	CPU	RealTime		CPU	RealTime
Proc Sql 1:	321 sec	344 sec	DataStep:	276 sec	278 sec
DataStep:	20 sec	38 sec			
Proc Sql 2:	32 sec	42 sec			
Summe:	373 sec	424 sec	**Summe:**	276 sec	278 sec

Die Zahlen zeigen, dass die Nutzung des Hash Objektes gerade bei großen Tabellen deutlich ressourcenschonender ist als die klassische Kombination von Proc Sql und

DataStep. Es konnten Einsparungen beim CPU Verbrauch von über 25 % erzielt werden. Auch beim realen Zeitverbrauch, der bei immer kürzeren Zeitfenstern für die Batchverarbeitung auch immer relevanter wird, konnten Einsparungen von bis deutlich über 30 % erzielt werden.

Dadurch dass die Hash Tabelle zu Beginn der Verarbeitung komplett in den Hauptspeicher geladen wird und danach sämtliche Lookups aus dem Hauptspeicher erledigt werden können, ist dieses Verfahren äußerst performant. Bei der Größe der Dimensionstabelle ist jedoch auch immer die Größe des Hauptspeichers zu beachten.

Im vorliegenden Beispiel enthält die Dimensionstabelle 4 numerische Variablen à 8 Byte und ca. 72.000 Datensätze. Es ergibt sich somit ein Speicherbedarf für das Hash Objekt von 4 x 72.000 x 8 Byte = 2.304.000 Byte, was ca. 2,2 MB entspricht und im vorliegenden Fall als unproblematisch einzustufen ist. Kommen jedoch alphanumerische Variablen größerer Länge ins Spiel, so kann der Hauptspeicher schnell zu klein werden.

2 Relationale Datenbanken mit SAS Base auslesen

Für den Zugriff auf relationale Datenbanken ist in der Regel die Lizenz eines SAS Access Moduls notwendig. Dies stellt die Möglichkeit bereit, über ein Libname Statement direkt auf die Tabellen innerhalb der relationalen Datenbank zuzugreifen.

```
01 LIBNAME MYLIB DB2 SSID=DB2ID AUTHID=DB2AUTH;
02 PROC PRINT DATA=MYLIB.TABELLE; RUN;
```

Im Libname Statement wird die Datenbank Engine (hier DB2) angegeben. Weiterhin werden die Zugriffsparameter wie Benutzer, Passwort, Tablespace, SSID oder AuthId mit angegeben, so dass der Zugriff technisch möglich ist. Den Transport der ausgelesenen Daten von der Datenbank in die SAS Ausführungsumgebung findet für den Anwender transparent statt.

Ist im Lizenzumfang kein Access Modul enthalten, ist es mit den Standard Mitteln nicht möglich, auf externe Datenbanksysteme zuzugreifen. In diesem Fall stehen nur die Standard-Engines, wie z.B. die SAS Base Engine oder die SAS SPD Engine zur Verfügung. Nachteil dieser Engines ist jedoch, dass die SAS Session beim Schreiben bzw. Aktualisieren von Daten die zugrundeliegenden Dateien exklusiv sperrt. Somit ist ein lesender oder schreibender Zugriff auf diese Tabellen durch andere SAS Programme während des Verarbeitungsvorgangs nicht mehr möglich. Die Sperre wird, je nach verwendeter SAS Engine, teilweise erst mit dem Schließen der jeweiligen SAS Session wieder aufgehoben.

In der Regel stellt dieses Sperrverhalten keine gravierende Einschränkung dar. Im Rahmen von ETL-Prozessen wird jedoch häufig ein Verarbeitungsframework genutzt, welches die einzelnen Verarbeitungsschritte des ETL-Prozesses in Tabellen protokolliert.

Jeder ETL-Prozess wird dabei in mehrere, nacheinander ablaufende Schritte unterteilt. Zum Start des ETL-Prozesses wird der Status des letzten Prozesslaufes aus der Datenbank ausgelesen. Nach jedem erfolgreich abgeschlossenem Schritt wird dieser in der Datenbank protokolliert.

Kommt es bei der Verarbeitung des ETL-Prozesses zu einem Fehler, so dass der Prozess vorzeitig beendet werden muss, so kann nach erfolgreicher Fehlerbehebung der Prozess an genau der Stelle des Abbruchs erneut gestartet werden. Die zuvor bereits fehlerfrei ausgeführten Prozessschritte müssen in diesem Fall nicht erneut ausgeführt werden, was zu kürzeren Laufzeiten und geringeren Prozesskosten führt.

In größeren ETL-Abläufen werden zur Performance-Steigerung häufig mehrere ETL-Prozesse parallel ausgeführt. Die Speicherung der Prozessergebnisse in SAS Tabellen unter Nutzung der SAS Base Engine oder der SPD Engine stößt hier schnell an ihre Grenzen, da jeder parallel laufende Prozess die Protokolltabelle exklusiv sperrt und somit eine parallele Verarbeitung faktisch unmöglich wird. Es bleibt nur die Möglichkeit der Speicherung der Protokolltabelle in einer relationalen Datenbank, da diese in der Regel auf Datensatzebene sperrt und nicht auf Tabellenebene.

Im Folgenden wird gezeigt, wie mit einem einfachen Trick auch mit einer reinen SAS Base Lizenz eine relationale Datenbank genutzt werden kann, um den Prozessstatus zu speichern. Hierzu wird das DataStep Component Interface to Java genutzt. Folgende Komponenten werden benötigt:

- Relationale Datenbank (hier: MySQL©)
- JDBC-Treiber für die genutzte Datenbank
- Java-Klasse für den Datenbankzugriff mit entsprechenden Zugriffsmethoden
- SAS Programm mit einem DataStep

2.1 Datenbank

Im hier gezeigten Beispiel wird eine einfache MySQL Datenbank mit einer einzelnen Protokolltabelle genutzt. Die Protokolltabelle hat dabei folgenden Aufbau:

Tabelle 2: Struktur der MySQL Protokolltabelle

Feld	Typ	Null	Key	Extra
ID	INT(11)	NO	PRI	AUTO_INCREMENT
PROGRAM	VARCHAR(30)	NO		
STATUS	INT(11)	NO		

Für jeden ETL Prozess wird ein Datensatz in der Tabelle erzeugt. Jedes Programm kann nur genau einen Datensatz in der Tabelle enthalten. Im Feld Status wird der Prozessstatus fortgeschrieben. Jedes Programm startet im Status 0. Nach erfolgreicher Prozessdurchführung wird die Stepnummer des ausgeführten Steps in der Datenbank gespeichert. Nach Abschluss des ETL-Prozesses wird der Status zur Dokumentation einer

fehlerfreien Durchführung wieder auf den Status 0 gesetzt, so dass beim nächsten ETL-Prozessdurchlauf die Verarbeitung wieder von vorne beginnen kann.

2.2 Java-Klasse für den Datenbankzugriff

Für den Datenbankzugriff wird eine Java Klasse mit zwei Methoden benötigt.

```
public double getStatusForProgram(String program, String host,
                  String database, String user, String password) {
  try {
    Class.forName("com.mysql.jdbc.Driver");
    connect = DriverManager.getConnection("jdbc:mysql://" + host +
        "/" + database + "?user=" + user + "&password=" + password);
    preparedStatement = connect.prepareStatement(
                      "SELECT status FROM log WHERE program = ?;");
    preparedStatement.setString(1, program);
    resultSet = preparedStatement.executeQuery();
    if (resultSet.first()) {
      int status = resultSet.getInt("status");
      resultSet.close();
      connect.close();
      return status;
    }
    return 0;
  } catch (Exception e) {
    System.out.println(e.toString());
  }
  return -1;
}
```

Die erste Methode liest den Status für das als Parameter übergebene Programm aus. Hierzu wird eine Verbindung zur Datenbank über den JDBC Connector geöffnet und ein SELECT auf die Log-Tabelle ausgeführt. Wird das gesuchte Programm in der Log-Tabelle gefunden, wird der Status zurückgeliefert. Wird das Programm nicht gefunden, wird der Status 0 zurückgeliefert. In diesem Fall wird das Programm erstmalig ausgeführt, und es existiert noch kein vorheriger Lauf. Tritt beim Aufbau der Verbindung zur Datenbank oder beim Abruf ein Fehler auf, so wird die Exception abgefangen und als Ausgabe auf die Konsole ausgegeben. In diesem Fall wird ein fester Status von -1 als Ergebnis zurückgeliefert.

```
public boolean setStatusForProgram(String program, double status,
       String host, String database, String user, String password) {
  try {
    Class.forName("com.mysql.jdbc.Driver");
    connect = DriverManager.getConnection("jdbc:mysql://" + host +
        "/" + database + "?user=" + user + "&password=" + password);
    preparedStatement = connect.prepareStatement(
                      "SELECT status FROM log WHERE program = ?;");
    preparedStatement.setString(1, program);
```

```
    resultSet = preparedStatement.executeQuery();
    if (resultSet.first()) {
      int oldstatus = resultSet.getInt("status");
      resultSet.close();
      preparedStatement = connect.prepareStatement(
                  "UPDATE log SET status = ? WHERE program = ?;");
      preparedStatement.setInt(1, new Double(status).intValue());
      preparedStatement.setString(2, program);
      preparedStatement.executeUpdate();
      connect.close();
      return true;
    }
    preparedStatement = connect.prepareStatement(
                "INSERT INTO log (program, status) VALUES (?, ?);");
    preparedStatement.setString(1, program);
    preparedStatement.setInt(2, new Double(status).intValue());
    preparedStatement.executeUpdate();
    connect.close();
    return true;
  } catch (Exception e) {
    System.out.println(e.toString());
  }
  return false;
}
```

Die zweite Methode aktualisiert den Status des Programms in der Datenbank. Zunächst wird der aktuelle Status aus der Datenbank ausgelesen. Existiert bereits ein Status, so wird dieser Datensatz aktualisiert, andernfalls wird ein neuer Datensatz eingefügt. Tritt bei den Datenbankoperationen ein Fehler auf, wird dieser abgefangen und auf der Konsole ausgegeben. War der Update-Vorgang erfolgreich, liefert die Methode den Wert true zurück, andernfalls false.

2.3 Nutzung der Java Klasse im DataStep

Java Klassen können innerhalb einer SAS Session in jedem Datastep genutzt werden. Damit die SAS Session die Klassen-/JAR-Dateien für die Ausführung findet, ist es erforderlich, beim Start den SAS Classpath korrekt zu belegen. Hierzu kann gut die SAS Konfigurationsdatei sasv9.cfg genutzt werden. Werden mehrere JAR-Dateien benötigt, können diese einfach hintereinander angegeben werden.

Ein beispielhafter Konfigurationsdateieintrag sieht wie folgt aus:

```
-set classpath "C:\KSFE\SASLOG.jar;C:\KSFE\MYSQL.jar"
```

Ein Datastep zum Auslesen des aktuellen Status, Ändern des Status und erneutem Auslesen des Status sieht wie folgt aus:

```
DATA _NULL_;
  DECLARE JAVAOBJ j;
  j = _NEW_ JAVAOBJ("de/viadee/sas/SasLog");
  j.exceptiondescribe(1);
```

```
  j.calldoublemethod("getStatusForProgram", "Prozess1", "localhost",
                     "sas", "sas", "sas", status);
  PUT status=;
    j.callBooleanMethod("setStatusForProgram", "Prozess1", 3,
                       "localhost", "sas", "sas", "sas", ergebnis);
  PUT ergebnis=;
  j.calldoublemethod("getStatusForProgram", "Prozess1", "localhost",
                    "sas", "sas", "sas", status);
  PUT status=;
RUN;
```

Folgende Ausgabe wird von dem DataStep produziert:

```
status=0
ergebnis=1
status=3
NOTE: DATA-Anweisung used (Total process time):
      real time           0.35 seconds
      cpu time            0.01 seconds
```

2.4 Fazit

Das kleine Beispiel hat gezeigt, wie einfach es ist, aus einem SAS DataStep heraus externe Java Methoden aufzurufen und auf die im Java Programm ermittelten Werte zuzugreifen.

Gerade für Einsatzgebiete, bei denen es nicht auf die Verarbeitung von großen Datenmengen ankommt, sondern nur auf die gleichzeitige Nutzung einer Tabelle durch mehrere Programme bietet dieses Verfahren eine kostengünstige Möglichkeit, auf relationale Datenbanken zuzugreifen.

Gleichzeitig bietet das Java Interface fast unendliche Möglichkeiten, auf externe Systeme aus einer SAS Verarbeitung heraus zuzugreifen.

Literatur

[1] R. Ray, J. Secosky: Better Hashing in SAS 9.2, Paper 308-08, http://support.sas.com/rnd/base/datastep/dot/better-hashing-sas92.pdf, Download am 16.02.2012.

[2] SAS: The Java Object and the DATA Step Component Interface, http://support.sas.com/rnd/base/datastep/dot/javaobj.html, Download am 16.02.2012