Geht nicht? Gibt's nicht! Daten lesen mit dem Data Step

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

Zusammenfassung

Wie bekomme ich Daten aus einer DB2-Datenbank in das SAS-System? Die gängige Antwort ist sicherlich die Nutzung der SAS/ACCESS to DB2-Schnittstelle. Was aber, wenn diese Schnittstelle nicht zur Verfügung steht? In diesem Vortrag wird ein alternativer Lösungsweg gezeigt, wie automatisiert DB2-Tabellen mit SAS importiert werden können. Fokus liegt hier auf der Erstellung eines Spiegelbestandes, da künftige Auswertungen auf den DB2-Bestand die produktiven Systeme nicht behindern oder beeinflussen dürfen.

Schlüsselwörter: Data Step, Datenschritt, IBM ixf-Format, DB2-Daten importieren

1 Motivation

Im Rahmen eines ETL-Projektes wurde aus den SAS-ETL-Prozessen heraus ein Zugriff auf ausgewählte DB2-Tabellen des produktiven OLTP-Systems erforderlich. Um die Unabhängigkeit der dispositiven ETL-Verarbeitung vom produktiven System zu wahren sollte im Rahmen einer nächtlichen Batchverarbeitung ein Spiegelbestand der benötigten DB2-Tabellen erstellt werden.

2 Mögliche Lösungswege

2.1 SAS/ACCESS to DB2 vom PC

Der triviale Ansatz wäre die Nutzung der SAS-Komponente ACCESS to DB2, um direkt auf die DB2-Tabellen zugreifen zu können. Da diese Komponente in der Zielumgebung jedoch nicht vorhanden ist, schied diese Möglichkeit direkt aus und es musste nach einer Alternative gesucht werden.

2.2 SAS/ACCESS to DB2 vom HOST

Da unter z/OS auch SAS zur Verfügung steht, wurde dies als zweite Alternative in Betracht gezogen. Hier besteht die Möglichkeit des direkten Zugriffs von SAS auf die DB2-Daten. Da die Zielumgebung jedoch eine dezentrale PC-Verarbeitung ist, müssten die Daten anschließend vom HOST auf den PC übertragen werden. Für diese Übertragung könnte SAS/CONNECT genutzt werden, dies stand jedoch auch nicht zur Verfü-

gung. Alternativ besteht die Möglichkeit, auf dem HOST die Tabellen mittels PROC CPORT einzupacken, anschließend auf den PC zu übertragen und dort mittels PROC CIMPORT wieder auszupacken.

Nachteil dieses Lösungsweges ist, dass ein Großteil der Verarbeitung auf dem zentralen HOST stattfindet. Die üblichen Verrechnungsmodelle sehen jedoch vor, dass HOST-Ressourcen entsprechend der CPU-Nutzung nach einem definierten Kostenschlüssel berechnet werden. Bei regelmäßiger Anwendung dieses Lösungsweges sind demzufolge nicht zu unterschätzende HOST-Kosten in die Analyse einzubeziehen.

2.3 Erstellung von CSV-Exporten mittels DB2-Client

Bei dieser Alternative wird ein lokaler DB2-Client genutzt, der sich mittels DB2- Gateway mit der HOST-Datenbank verbindet. Über DB2-Client Utilities besteht die Möglichkeit, einzelne Tabellen in unabhängige Formate, wie z.B. CSV-Dateien zu exportieren.

Nachteil dieser Lösung ist, dass das CSV-Format nicht für alle Datensituationen robust genug ist. Es muss ein Trennzeichen definiert werden, welches nach Möglichkeit in den Daten der Tabelle nicht vorkommt. Weiterhin muss sichergestellt sein, dass Zeilenumbrüche oder ausgefallene Sonderzeichen in den Daten nicht vorkommen, da diese ein nachgelagertes Einlesen der Daten erschweren würden.

Gravierender ist jedoch, dass das CSV-Format nicht selbsterklärend ist. Für die Verarbeitung der Dateien mittels SAS muss immer ein Einleseprogramm gepflegt werden, welches die Datensatzbeschreibung kennt und die Daten entsprechend einliest. Bei Änderungen an der DB2-Tabelle muss zwingend auch das Einleseprogramm angepasst werden, damit die jeweiligen Spalten korrekt eingelesen werden können.

2.4 PC-Version des ixf-Dateiformats

Beim ixf-Dateiformat handelt es sich um ein proprietäres IBM-Dateiformat, welches seitens IBM genutzt wird, um Tabellen zwischen verschiedenen Datenbanken auszutauschen. ixf-Dateien werden mit Hilfe des DB2-Clients erstellt und enthalten alle Informationen über die exportierte Tabelle. Dies bedeutet, dass neben den Tabellendaten auch die Tabellenstruktur und die Tabellenmetadaten wie z.B. Primärschlüssel in dem Dateiformat enthalten sind.

Ein weiterer Vorteil ist, dass auch beim Zugriff auf eine HOST DB2-Datenbank der größere Anteil am CPU-Verbrauch auf dem Client stattfindet. Sämtliche Zeichensatzkonvertierungen werden vom DB2-Client ausgeführt; der HOST liefert lediglich die Daten aus der Datenbank. Hieraus folgt, dass bei der Nutzung dieses Formates nur mit einem überschaubaren Anstieg der HOST-Kosten zu rechnen ist.

Einziger Nachteil ist, dass es sich bei dem Format um ein proprietäres Format handelt, welches jedoch glücklicherweise gut dokumentiert ist [1].

Da SAS den Data Step immer als Universalwerkzeug zum Einlesen von Daten darstellt, soll im Folgenden gezeigt werden, wie sich selbst dieses proprietäre Format problemlos mit SAS einlesen lässt und so ein automatisierter Prozess zum Import von DB2- Tabellen entsteht.

3 Umsetzung

3.1 Beschreibung des Lösungsweges

Der Prozess zur automatischen Übernahme einer DB2-Tabelle sieht vor, dass aus einer SAS-Session heraus der DB2-Client zur Erstellung der ixf-Transportdatei gestartet wird. Der Speicherort der Transportdatei wird automatisch im Ordner einer temporären Bibliothek gewählt.

Nach der Erstellung der Transportdatei wird diese automatisch eingelesen und in eine SAS-Tabelle konvertiert. Bei der Konvertierung bleiben die Spaltennamen, Datentypen sowie in der DB2-Tabelle vorhandene Indizes erhalten.

Um Transferzeiten zu sparen muss die Möglichkeit einer selektiven Übernahme von aktualisierten Datensätzen bestehen. Hierzu müssen die geänderten Datensätze in der DB2-Tabelle anhand eines Aktualisierungszeitstempels erkennbar sein. Weiterhin muss sichergestellt sein, dass Löschungen von Datensätzen nicht vorkommen bzw. dass gelöschte Datensätze separat übermittelt und verarbeitet werden. Bei einem Delta-Transfer werden nur die Datensätze berücksichtigt, deren Aktualisierungszeitstempel aktueller als der Zeitpunkt des letzten Datenabzuges ist.

Handelt es sich bei der Verarbeitung um eine Delta-Verarbeitung, so muss der aktuelle Datenabzug mit dem vorherigen Datenabzug kombiniert werden. Neue Sätze müssen angefügt werden, geänderte Sätze müssen ausgetauscht werden.

Aus dieser Prozessbeschreibung ergeben sich folgende Punkte, auf die im Folgenden näher eingegangen werden soll:

- Start von Sub-Prozessen (DB2-Client) und Überwachung des Return-Codes
- Lesen des IBM ixf-Formates
- Erstellen von Indizes
- Zusammenführen von Daten

3.2 Start von DB2-Sub-Prozessen

Um aus einer SAS-Session heraus mit dem DB2-Client zu kommunizieren und Extraktionsprozesse anzustoßen, ist es notwendig, dass die SAS-Session für die Nutzung von

x-Commands freigeschaltet ist. Hierzu ist ggf. eine Anpassung der SAS-Konfiguration notwendig, da bei einer Umgebung mit dem SAS-Metadatenserver standardmäßig x-Commands deaktiviert sind.

Für den Entladeprozess werden ein temporäres Verzeichnis und diverse temporäre Dateien benötigt. Weiterhin werden genauere Infos zu der Tabelle, den Spalten und dem abzugreifenden Zeitraum benötigt. Alle diese Infos werden in Makrovariablen gespeichert:

```
%LET DB2USER   = MyUser;
%LET DB2PASS   = MyPassword;
%LET DB2DB     = MyDB;
%LET DB2SCHEMA = MySchema;
%LET DB2TAB    = MyTabelle;
%LET DB2AEND   = MyAendZstCol;
%LET DB2AEND_V = '2013-01-01-00.00.00.000000';
%LET DB2AEND_B = '2013-02-01-00.00.00.000000';

%LET DB2BASIS  = E:\TEMP\;
%LET DB2SQL    = &DB2BASIS.db2.sql;
%LET DB2CMD    = &DB2BASIS.db2.cmd;
%LET DB2IXF    = &DB2BASIS.db2.ixf;
%LET DB2LOG    = &DB2BASIS.db2.log;
%LET DB2MSG    = &DB2BASIS.db2.msg;
```

Zunächst muss das SQL generiert werden, mit dem der DB2-Client die Entladung der DB2-Tabelle vornimmt:

```
FILENAME db2sql "&DB2SQL.";
FILENAME db2cmd "&DB2CMD.";

DATA _NULL_;
 FILE db2sql;
 PUT "CONNECT TO &DB2DB. USER &DB2USER. USING &DB2PASS.";
 PUT ";";
 PUT "EXPORT TO &DB2IXF. OF IXF ";
 PUT " MESSAGES &DB2MSG.";
 PUT "  SELECT A.*";
 PUT "  FROM &DB2SCHEMA..&DB2TAB. A";
 PUT "  WHERE A.&DB2AEND >= &DB2AEND_V. ";
 PUT "    AND A.&DB2AEND <  &DB2AEND_B. ";
 PUT "  FOR FETCH ONLY WITH UR";
 PUT ";";
RUN;
```

Anschließend wird der Aufruf des DB2-Command Line Interfaces generiert:

```
DATA _NULL_;
 FILE db2cmd;
 PUT "DB2CMD /C /W DB2 -c -ec -t -v -w -f &db2sql -z &db2log.";
 PUT 'SET DB2RC=%ERRORLEVEL%';
 PUT 'EXIT /B %DB2RC%';
RUN;
```

Zu beachten ist, dass über DB2CMD ein DB2-Command Line Interface (CLI) aufgerufen wird, dem der Befehl DB2 inkl. weiterer Parameter übergeben wird. Über die Schalter /C /W wird dem DB2-CLI mitgeteilt, auf die Beendigung des Befehls DB2 zu warten und anschließend das CLI zu beenden. Der Return-Code des CLI wird in einer Umgebungsvariablen gespeichert und anschließend an den aufrufenden Prozess zurückgegeben.

Zum Abschluss wird das generierte Skript über einen SYSTASK COMMAND aufgerufen.

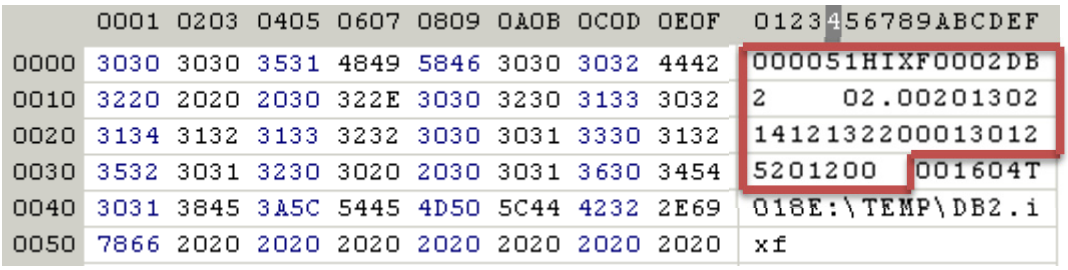
```
DATA _NULL_;
 SYSTASK COMMAND "CALL &db2cmd." MNAME=DB2TASK STATUS=DB2RC;
 WAITFOR _ALL_ &DB2TASK. TIMEOUT=60;
 PUT "DB2RC = &DB2RC.";
RUN;
```

Standardmäßig läuft der Data Step nach dem Start des Systasks weiter, so dass mehrere Systasks parallel laufen können. Über den WAITFOR Befehl wird dann unter Angabe eines Timeouts auf alle gestarteten Prozesse gewartet. Der Name der gestarteten Prozesse wird in der unter MNAME= angegeben Makrovariablen gespeichert, der Status in der unter STATUS= angegeben Makrovariablen.

Über diese Variablen kann der Erfolg des Entladevorgangs überwacht und die weitere Steuerung des Datenabzugs vorgenommen werden.

3.3 Lesen des ixf-Formates – Metadaten

Schaut man sich die erzeugte ixf-Datei mit einem Editor an, so stellt man schnell fest, dass es sich hierbei zwar um ein binäres Format handelt, jedoch viele Tabelleninfos direkt auslesbar sind.



	0001	0203	0405	0607	0809	0A0B	0C0D	0E0F	0123456789ABCDEF
0000	3030	3030	3531	4849	5846	3030	3032	4442	000051HIXF0002DB
0010	3220	2020	2030	322E	3030	3230	3133	3032	2 02.00201302
0020	3134	3132	3133	3232	3030	3031	3330	3132	14121322000130 12
0030	3532	3031	3230	3020	2030	3031	3630	3454	5201200 001604T
0040	3031	3845	3A5C	5445	4D50	5C44	4232	2E69	018E:\TEMP\DB2.i
0050	7866	2020	2020	2020	2020	2020	2020	2020	xf

Abbildung 1: Auszug ixf-Datei

Das ixf-Format gliedert sich in folgende Satztypen:

- H-Satz (Kopfsatz)
- T-Satz (Tabellensatz)
- C-Satz (Spaltendeskriptorsatz)
- D-Satz (Datensatz)
- A-Satz (Anwendungssatz)

Jeder Satz ist so aufgebaut, dass die ersten 6 Byte die Länge des jeweiligen Satzes angeben. Anschließend kommt 1 Byte für die Angabe des Satztypes. (vgl. Abb. 1: 51 Byte für den H-Satz und 1.604 Byte für den T-Satz).

Aus den H-/T-/C-Sätzen können alle Informationen gewonnen werden, die die übertragene Tabelle beschreiben. Hierzu zählen folgende Informationen:

- H-Satz:
  - Datum und Uhrzeit des Exportes
- T-Satz:
  - Tabellenname
  - Anzahl Spaltendeskriptoren
  - Tabellenbeschreibung
- C-Satz:
  - Spaltenname
  - Spaltenlänge
  - Datentyp
  - D-Satz-Kennung

In einem ersten Schritt wird über einen Data Step die ixf-Datei eingelesen, wobei hier nur die H-/T- und C-Sätze gelesen werden. Weiterhin wird die Position des ersten D-Satzes ermittelt, damit später direkt zu dieser Stelle gesprungen werden kann.

Da es sich bei der ixf-Datei um ein Binärformat handelt, welches keine CR-/LF-Zeichen enthält, ist es erforderlich, dass die Datei im Filename Statement mit dem RecFM=N (Binary Format) eingelesen wird.

```
FILENAME IMPORT "&DB2IXF." RECFM=F;

DATA _NULL_;
 INFILE IMPORT;
 ATTRIB INPUT_FORMAT FORMAT=$40.;
 ATTRIB OUTPUT_FORMAT FORMAT=$40.;
 ATTRIB NULL_VAL FORMAT=$2.;
 RETAIN MAX_D_SATZ 0;

 INPUT H_LEN 6. @;

 INPUT @27 H_DATUM_J 4.
       @31 H_DATUM_M 2.
       @33 H_DATUM_T 2. @;
 H_DATUM = MDY(H_DATUM_M, H_DATUM_T, H_DATUM_J);
 CALL SYMPUT("H_DATUM", STRIP(PUT(H_DATUM, 8.)));

INPUT @(H_LEN + 6 + 1)   T_LEN              6.
      @(H_LEN + 6 + 8)   T_NAME_LEN         3.
      @(H_LEN + 6 + 11)  T_NAME             $VARYING256. T_NAME_LEN
      @(H_LEN + 6 + 546) T_DAT_ANZ_C_SATZ   5.
      @(H_LEN + 6 + 553) T_DAT_BESCHREIBUNG $30. @;

 CALL SYMPUT("TAB_SPALTEN", STRIP(PUT(T_DAT_ANZ_C_SATZ, 8.)));
 C_LEN_SUM = 0;

 DO I=1 TO T_DAT_ANZ_C_SATZ;
  INPUT @(H_LEN+6+T_LEN+6+C_LEN_SUM+1)   C_LEN                    6.
        @(H_LEN+6+T_LEN+6+C_LEN_SUM+8)   C_NAME_L                 3.
        @(H_LEN+6+T_LEN+6+C_LEN_SUM+11)  C_NAME
                                             $VARYING256. C_NAME_L
        @(H_LEN+6+T_LEN+6+C_LEN_SUM+267) C_NULL                  $1.
        @(H_LEN+6+T_LEN+6+C_LEN_SUM+272) C_DATACLASS             $1.
        @(H_LEN+6+T_LEN+6+C_LEN_SUM+273) C_DATATYPE               3.
        @(H_LEN+6+T_LEN+6+C_LEN_SUM+286) C_COL_LEN               $5.
        @(H_LEN+6+T_LEN+6+C_LEN_SUM+291) C_SATZKENNUNG_D          3.
        @(H_LEN+6+T_LEN+6+C_LEN_SUM+294) C_SPALTENPOSITION        6.
        @(H_LEN+6+T_LEN+6+C_LEN_SUM+300) C_SPALTENBESCHREIBUNG $30.
        @;

  MAX_D_SATZ = MAX(MAX_D_SATZ, C_SATZKENNUNG_D);
  C_LEN_SUM + C_LEN + 6;

  SELECT (C_DATATYPE);
```

```
WHEN (384) DO; /* DATE */
 INPUT_FORMAT = "DB2DATUM.";
 OUTPUT_FORMAT = "DATE9.";
 NULL_VAL = "''";
END;
WHEN (388) DO; /* TIME */
 INPUT_FORMAT = "TIME8.";
 OUTPUT_FORMAT = "TIME8.";
 NULL_VAL = ".";
END;
WHEN (392) DO; /* TIMESTAMP */
 INPUT_FORMAT = "DB2TIMESTAMP.";
 OUTPUT_FORMAT = "DATETIME30.6";
 NULL_VAL = ".";
END;
WHEN (452) DO; /*CHAR */
 INPUT_FORMAT = "$CHAR"!!STRIP(PUT(INPUT(C_COL_LEN,5.),5.))!!".";
 OUTPUT_FORMAT = "$"!!STRIP(PUT(INPUT(C_COL_LEN,5.),5.))!!".";
 NULL_VAL = "''";
END;
WHEN (448) DO; /* VARCHAR */
 INPUT_FORMAT = "VARCHAR";
 OUTPUT_FORMAT = "$"!!STRIP(PUT(INPUT(C_COL_LEN,5.),5.))!!".";
 NULL_VAL = "''";
END;
WHEN (500) DO; /* SMALLINT */
 INPUT_FORMAT = "IB2.";
 OUTPUT_FORMAT = "6.";
 NULL_VAL = ".";
END;
WHEN (496) DO; /* INTEGER */
 INPUT_FORMAT = "IB4.";
 OUTPUT_FORMAT = "11.";
 NULL_VAL = ".";
END;
WHEN (492) DO; /* BIGINT */
 INPUT_FORMAT = "IB8.";
 OUTPUT_FORMAT = "32.";
 NULL_VAL = ".";
END;
WHEN (484) DO; /* DECIMAL */
 INPUT_FORMAT = "S370FPD"!!
      STRIP(PUT(CEIL((INPUT(SUBSTR(C_COL_LEN,1,3),3.)+1)/2),3.))
              !!"." !!
```

```
          STRIP(PUT(INPUT(SUBSTR(C_COL_LEN,4,2),2.),2.));
     OUTPUT_FORMAT = STRIP(PUT(MIN(INPUT(SUBSTR(C_COL_LEN,1,3),3.)
           +2,32),3.))!!"."!!STRIP(PUT(INPUT(SUBSTR(C_COL_LEN,4,2)
           ,2.),2.));
     NULL_VAL = ".";
    END;
    OTHERWISE DO;
     PUTLOG "ACHTUNG: Datentyp nicht definiert!";
     PUT _All_;
    END;
   END;

   CALL SYMPUT(COMPRESS("COL_NAME_"!!PUT(I,8.)), STRIP(C_NAME));
   CALL SYMPUT(COMPRESS("COL_DSATZ_"!!PUT(I,8.)),
                                 STRIP(PUT(C_SATZKENNUNG_D, 8.)));
   CALL SYMPUT(COMPRESS("COL_NULL_"!!PUT(I, 8.)), STRIP(C_NULL));
   CALL SYMPUT(COMPRESS("COL_NULLVAL_"!!PUT(I, 8.)),
                                 STRIP(NULL_VAL));
   CALL SYMPUT(COMPRESS("COL_DATATYPE_"!!PUT(I, 8.)),
                                 STRIP(PUT(C_DATATYPE, 8.)));
   CALL SYMPUT(COMPRESS("COL_LEN_"!!PUT(I, 8.)), STRIP(C_COL_LEN));
   CALL SYMPUT(COMPRESS("COL_IFRM_"!!PUT(I, 8.)),
                                 STRIP(INPUT_FORMAT));
   CALL SYMPUT(COMPRESS("COL_OFRM_"!!PUT(I, 8.)),
                                 STRIP(OUTPUT_FORMAT));
   CALL SYMPUT(COMPRESS("COL_POS_"!!PUT(I, 8.)),
                                 STRIP(PUT(C_SPALTENPOSITION, 8.)));
 END;

 D_START_POS = H_LEN + 6 + T_LEN + 6 + C_LEN_SUM;
 CALL SYMPUT("START_D", STRIP(PUT(D_START_POS, 20.)));
 CALL SYMPUT("MAX_D_SATZ", STRIP(PUT(MAX_D_SATZ, 20.)));
 STOP;
RUN;
```

In dem Data Step werden verschiedenste Methoden zum Einlesen von Daten genutzt. Die gebräuchlichste davon ist

```
  INPUT @Position Variable Einleseformat. (@)
```

Hierbei wird von der aktuellen Position des Einlesecursors die Anzahl an Zeichen, die unter @Position angegeben ist, nach rechts weitergesprungen. Dann wird die Anzahl an Zeichen, die sich aus der Definition des Einleseformats ergibt, eingelesen und in der

angegebenen Variablen gespeichert. Das letzte @ Zeichen hält den Einlesecursor fest, so dass im selben Datensatz ein weiteres INPUT-Statement folgen kann.

Da bei ixf-Dateien die Satzlänge eines Satzes zu Beginn des Satzes angegeben ist, muss die Position, von der eingelesen wird, dynamisch ermittelt werden. Hierzu besteht die Möglichkeit, die Position anhand einer Formel aus Zahlen und Data Step Variablen zu berechnen. Es ergibt sich somit folgendes INPUT-Statement

```
INPUT @(A + B) Variable Einleseformat. (@)
```

In diesem Fall wird zunächst die Formel berechnet um den Offset zu ermitteln.

Eine weitere Besonderheit ist das Einlesen der Spalten-/Tabellennamen. Diese können bis zu einer Länge von 256 Zeichen angegeben werden; die Länge des Spalten-/Tabellennamens wird in einem separaten Feld angegeben. Um diese Werte korrekt einlesen zu können wird das Format VARYING genutzt:

```
INPUT @Position Variable $VARYING256. VarLaenge (@)
```

In diesem Fall werden bis zu 256 Zeichen eingelesen, es werden aber nur die in der Variablen VarLaenge angegebene Anzahl an Zeichen genutzt.

Alle Metainformationen über die zu importierende Tabelle werden in Makrovariablen gespeichert. Insbesondere wird die Position des ersten D-Satzes, der sich aus der Summe der Satzlängen der H-, T- und C-Sätze ergibt, gespeichert, damit im Folgenden zum Einlesen direkt an diese Position gesprungen werden kann. Weiterhin wird die maximale D-Satz-Nummer gespeichert. Lt. Definition kann ein einzelner D-Satz nur eine Länge von 32.771 Bytes haben. Hat die Tabelle längere Spalten, so werden die Spalten auf mehrere D-Sätze aufgeteilt.

Ein weiterer wichtiger Punkt bei der Verarbeitung der Metadaten ist die Übersetzung der DB2-Datenformate in SAS-Einlese- und -Ausgabeformate. Für viele DB2- Datentypen ist die Zuordnung des SAS-Formates trivial. Dennoch sind Kleinigkeiten zu beachten. So sind Datensätze vom Typ CHAR auch mit einem $CHAR-Format einzulesen, da nur so bei SAS auch führende Leerzeichen in den Datensätzen erhalten bleiben.

Besonderes Augenmerk verdienen die numerischen Formate. Lt. Definition werden Zahlen im Maschienenformat von IBM gespeichert. Dies entspricht nicht der ASCII Kodierung und somit sind die gebräuchlichen PC-Einleseformate von SAS für Zahlen nicht nützlich. Für ganzzahlige Werte sind die IB-Formate zu nutzen, da diese die Binärwerte korrekt in SAS-Zahlen übersetzen. Für Zahlen vom Datentyp DECIMAL ist das Format S370FBx.y zu nutzen, da diese Zahlen in der ixf-Datei im HOST-üblichen PackedDecimal-Format vorliegen. Aus der Formatlänge sind der Vorkomma- und der

Nachkommateil zu extrahieren. Weiterhin muss die Halbbyte-Zählung von IBM korrekt in die Vollbyte-Zählung von SAS umgerechnet werden.

Dies bedeutet, dass eine Zahl mit der ixf-Länge 01302 genau dreizehn Ziffern enthält und davon 2 Nachkommastellen genutzt werden. Diese Zahl würde dem SAS-Ausgabeformat 13.2 entsprechen. Da die Zahl in der ixf-Datei jedoch PackedDecimal gespeichert wird, muss dieser Wert in das entsprechende PD7.2-Format umgerechnet werden. Dies geschieht über ganzzahliges Teilen, Runden und Zusammensetzen der Formatteile.

Eine weitere Besonderheit gibt es beim Einlesen von DB2-Datums-/Timestamp-Werten. Diese können in der DB2 für ein Datum im Zeitraum von 0001-01-01 bis 9999-12-31 und für einen Timestamp im Zeitraum 0001-01-01-00.00.00.000000 bis 9999-12-13-23.59.59.999999 variiert werden. Die Zeitraumobergrenze ist für SAS unproblematisch. Lediglich bei der Zeitraumuntergrenze kennt SAS-Datumswerte vor dem 01.01.1582 nicht.

Über ein spezielles Einleseformat kann dieses Problem umgangen werden.

```
PROC FORMAT;
  INVALUE DB2DATUM '0001-01-01' - < '1582-01-01' = .
                                          OTHER = (|YYMMDD10.|);
  INVALUE DB2TIMESTAMP '0001-01-01-00.00.00.000000' - <
                       '1582-01-01-00.00.00.000000' = .
                                          OTHER = (|YMDDTTM26.6|);
```

Über diese beiden Formate werden Datumswerte vor dem 01.01.1582 in SAS auf MISSING VALUE gesetzt. Alle anderen Datumswerte werden korrekt als SAS-Datum importiert. Warnungen oder Fehler werden im Log nicht angezeigt.

3.4 Lesen des ixf-Formates – Werte

Im zweiten Schritt wird die Ausgabetabelle erzeugt. Es werden genau die Spalten übernommen, die aus den Tabellenmetadaten ermittelt wurden. Für alle zu übernehmenden Spalten werden die im ersten Schritt ermittelten Formate, Einleseformate und Label vergeben.

```
DATA TEMP.TABELLE(KEEP=
                  %DO I=1 %TO &TAB_SPALTEN.;
                        &&COL_NAME_&I..
                  %END;
                  );

 %DO I=1 %TO &TAB_SPALTEN.;
   ATTRIB &&COL_NAME_&I.. FORMAT=&&COL_OFRM_&I..
```

```
                          INFORMAT=&&COL_OFRM_&I..
                          LABEL="&&COL_NAME_&I..";
%END;

INFILE IMPORT;

START_POS = &START_D.;

INPUT @(START_POS + 1) D_LEN 6.
      @(START_POS + 7) D_TYP $1.
      @;

DO WHILE (D_TYP = "D");

 INPUT @(START_POS + 8) D_KENNUNG 3.
       @;

 %DO I=1 %TO &TAB_SPALTEN.;
  IF &&COL_DSATZ_&I.. EQ D_KENNUNG THEN DO;
   %IF "&&COL_NULL_&I.." = "N" %THEN %DO;
    %IF "&&COL_IFRM_&I.." = "VARCHAR" %THEN %DO;
     INPUT @(START_POS+14+&&COL_POS_&I..) &&COL_NAME_&I.._Y IB2. @;
     INPUT @(START_POS+14+&&COL_POS_&I..+2) &&COL_NAME_&I..
                             $VARYING254. &&COL_NAME_&I.._Y @;
    %END;
    %ELSE %DO;
     INPUT @(START_POS+14+&&COL_POS_&I..) &&COL_NAME_&I..
                                          &&COL_IFRM_&I..  @;
    %END;
   %END;
   %ELSE %DO;
    INPUT @(START_POS+14+&&COL_POS_&I..) &&COL_NAME_&I.._X $2. @;
    IF &&COL_NAME_&I.._X EQ '0000'X THEN DO;
     %IF "&&COL_IFRM_&I.." = "VARCHAR" %THEN %DO;
      INPUT @(START_POS+14+&&COL_POS_&I..+2) &&COL_NAME_&I.._Y IB2. @;
      INPUT @(START_POS+14+&&COL_POS_&I..+4) &&COL_NAME_&I..
                              $VARYING254. &&COL_NAME_&I.._Y @;
     %END;
     %ELSE %DO;
      INPUT @(START_POS+14+&&COL_POS_&I..+2) &&COL_NAME_&I..
                                          &&COL_IFRM_&I.. @;
     %END;
    END;
    ELSE DO;
```

```
        &&COL_NAME_&I.. = &&COL_NULLVAL_&I..;
      END;
     %END;
    END;
   %END;

   IF D_KENNUNG EQ &MAX_D_SATZ. THEN OUTPUT;

   START_POS + D_LEN + 6;
   INPUT @(START_POS + 1) D_LEN 6.
         @(START_POS + 7) D_TYP $1.
         @;
 END;
 STOP;
RUN;
```

Über die DO-Schleife werden solange Datensätze verarbeitet, bis keine weiteren D-Sätze mehr vorhanden sind. Da ggf. ein Datensatz aus mehreren D-Sätzen bestehen kann, wird für jeden D-Satz die Satznummer geprüft. Es werden je D-Satz nur die zu der Satznummer gültigen Spalten verarbeitet und es wird jeweils nur nach dem letzten D-Satz ein Ausgabesatz produziert.

Beim Einlesen der Tabellendaten ergibt sich eine Besonderheit aus der Verarbeitung von NULL-Werten. Wenn eine Spalte NULL-Werte enthalten darf, müssen die ersten zwei Bytes des Spaltenwertes geprüft werden, ob diese den Wert ‚0000'X haben. In diesem Fall ist der Wert belegt, andernfalls müsste der Wert MISSING vergeben werden.

Bei VARCHAR-Werten muss weiterhin beim Einlesen die Länge des Wertes geprüft werden. Diese Länge steht wiederum in den ersten beiden Bytes des Spaltenwertes. Wie bereits in Kapitel 3.3 beschrieben wird das Einleseformat VARYING genutzt, um die korrekte Länge einzulesen.

3.5 Erstellen von Indizes

Beim Einlesen der Metadaten zu einer Tabelle kann je Spalte die Position im Primärschlüssel mit ausgelesen werden. Dies setzt voraus, dass ein Primärschlüssel in der Datenbank definiert ist.

Ist dies der Fall, so kann der Primärschlüssel automatisch bei der Erzeugung der Tabelle mit angelegt werden, ohne dass hierzu ein zusätzlicher PROC DATASETS- oder PROC SQL-Schritt benötigt wird.

Hierzu wird lediglich an die Tabellenausgabe neben dem KEEP-Statement ein INDEX-Statement angehängt, welches den Index erzeugt.

```
DATA TEMP.TABELLE(KEEP=
                  %DO I=1 %TO &TAB_SPALTEN.;
                        &&COL_NAME_&I..
                  %END;
                  %IF &ANZ_KEY_COL = 1 %THEN %DO;
                    INDEX=(&KEY_COL_1)
                  %END;
                  %ELSE %DO;
                    INDEX=(PK(
                            %DO I=1 %TO &ANZ_KEY_COL.
                              &&KEY_COL_&I..
                            %END;
                           ))
                  %END;
                  );
```

Dieses Verfahren setzt voraus, dass die Anzahl der Schlüsselspalten bekannt ist. Ist nur eine Schlüsselspalte vorhanden, so muss der Indexname mit dem Spaltennamen übereinstimmen. Liegen mehrere Indexspalten vor, so kann der Name frei gewählt werden.

3.6 Zusammenführen von Daten

Wird beim Erstellen der ixf-Datei nur ein Teilbereich mit den geänderten Datensätzen in der ixf-Datei berücksichtigt, so muss dieser Teil später mit dem Gesamtbestand zusammengefügt werden.

Auf den ersten Blick erscheint ein MERGE hier das Medium der Wahl. da sowohl der vorhandene Gesamtbestand, als auch der aus der ixf-Datei erzeugte Teilbestand über den Primärschlüssel indiziert sind und dieser somit wunderbar für den Merge genutzt werden kann.

Bei großen Datenmengen erweist sich dieses Vorgehen jedoch als äußerst inperformant. Beim Einfügen von 50.000 geänderten Datensätzen anhand des Primärschlüssels in eine Grundtabelle mit 82 Millionen Datensätzen hat der Merge weit über 10 Stunden Echtzeit benötigt. Für eine tägliche Datenaktualisierung ist dieser Wert nicht akzeptabel.

Eine Alternative stellt hier das Hash-Objekt dar. Dieses kann aufgrund der geringen Menge an geänderten Datensätzen problemlos genutzt werden. In einem DataStep werden die Originaltabelle und die Änderungstabelle mittels SET verbunden. Über die IN= Option werden die Datensätze nach ihrer Herkunft identifizierbar gemacht.

```
DATA ZIEL.TABELLE;
  SET ZIEL.TABELLE(IN=A)
      TEMP.TABELLE(IN=B);
  IF _N_ = 1 THEN DO;
    DECLARE HASH H(DATASET: "TEMP.TABELLE");
                %DO I=1 %TO %ANZ_KEY_COL;
                  H.DEFINEKEY("&&KEY_COL_&I..");
                %END;
                H.DEFINEDONE();
  END;
  IF A AND H.FIND() EQ 0 THEN DELETE;
RUN;
```

Anschließend wird ein Hash-Objekt auf die Änderungstabelle mit den Feldern des Primärschlüssels definiert. Über die letzte Zeile im Data Step wird der „Merge“ vorgenommen. In dem Fall, dass der Datensatz aus der Originaltabelle kommt und gleichzeitig der Schlüssel in der Hash-Tabelle gefunden wird, wird dieser Satz gelöscht. Alle anderen Sätze werden übernommen und die neuen, geänderten Sätze automatisch angefügt.

Angewandt auf dieselben Testdaten (50.000 Änderungssätze, 82 Millionen Originalsätze) liefert diese Variante bereits nach 15 Echtzeitminuten (5 Minuten CPU-Nutzung) das Ergebnis.

4. Fazit

Das Beispiel zeigt, dass der Data Step wirklich als Universalwerkzeug angesehen werden kann. In der Praxis hat sich gezeigt, dass über viele Monate zuverlässig ixf-Dateien mittels Data Step performant importiert werden können.

Betrachtet man die vielen Möglichkeiten, wie ein INPUT-Statement formuliert werden kann, so lassen sich fast beliebige, strukturierte Datenquellen damit einlesen und verarbeiten.

Dennoch ist es ratsam, immer ein Auge auf die Performance zu legen. Die Nutzung des MERGE zum Zusammenführen von Daten hat gezeigt, dass der Standardweg nicht immer der schnellste ist. Hier zeigt sich, dass die sukzessiv eingeführten Neuerungen im Data Step ihre Berechtigung haben. An den richtigen Stellen eingesetzt lässt sich damit die Verarbeitung deutlich optimieren.

Literatur

[1] Dokumentation des PC/IXF-Formates,
http://pic.dhe.ibm.com/infocenter/db2luw/v10r1/topic/com.ibm.db2.luw.admin.dm.doc/doc/r0004667.html
[13.02.2013]