Einsichten über (An)Sichten

Andreas Menrath
HMS Analytical Software GmbH
Rohrbacher Str. 26
69115 Heidelberg
Andreas.Menrath@analytical-software.de

Zusammenfassung

SAS Views gehören zu den am meisten unterschätzten Features der SAS/BASE Programmierung und bieten wesentlich mehr als nur eine vordefinierte Sicht auf bestehende Tabellen. Dieser Beitrag stellt die technischen Grundlagen vor und erläutert die jeweiligen Vor- und Nachteile. Der Fokus liegt jedoch auf der Vorstellung von Praxisbeispielen mit innovativen Lösungsansätzen, die den Mehrwert von Views im Entwickleralltag unterstreichen sollen.

Schlüsselwörter: DataStep View, SQL View, Dictionary-Tabellen, Webservice, Smart Prompts

1 Einleitung

SAS Views (bzw. im deutschsprachigen Gebrauch auch Ansichten genannt) sind eigentlich ein alter Hut. Als Dateien mit der Endung .sas7bvew fristen sie relativ unbeachtet ihr Nischendasein im Dateisystem und spielen fast ausschließlich im Kontext von Performance-Tuning-Maßnahmen eine nennenswerte Rolle.
Nach Auffassung des Autors vollkommen zu Unrecht! Views können nämlich noch wesentlich mehr, wenn man weiß wie man sie richtig ausreizen kann.

Dieser Beitrag erläutert die theoretischen Grundlagen und bietet eine Auswahl an praxisrelevanten Beispielen, in denen Views ihre volle Stärke ausspielen können.

2 Überblick

Obwohl SAS Views eher zu den fortgeschrittenen SAS-Techniken gehören, hat wohl schon so gut wie jeder Entwickler mit ihnen gearbeitet. Vielleicht hat sich der Leser aber auch schon einmal gefragt, was eine View von einer Tabelle unterscheidet und wie sie sich erzeugen lassen.
Im Folgenden werden noch einmal die Grundlagen kurz zusammengefasst.

2.1 Definition

Am einfachsten ist es, sich eine View als eine gespeicherte Abfrage vorzustellen. Im Gegensatz zu Tabellen enthalten Ansichten keine Daten, sondern nur Informationen über die Datenstruktur der Ausgabedaten (Spaltenname, Länge, Typ, usw.) sowie den Algorithmus zu ihrer Datenbereitstellung. Hieraus ergibt sich eine Reihe von Vor- und Nachteilen.

2.2 Vor- und Nachteile

Zu den Vorteilen gehören:

- Da Views keine Daten enthalten, kann Festplattenspeicherplatz eingespart werden.
- Daten, die durch eine View bereitgestellt werden, sind immer aktuell und Datenreplikation kann hierdurch verringert werden. Bei einem Datenabzug als Tabelle muss der Datenexport wiederholt werden, wenn sich die Quellsystemdaten geändert haben.
- Durch die Definition mehrerer Views können mehrere unterschiedliche Sichten auf die gleichen Quelldaten bereitgestellt werden, z.B. mit unterschiedlicher fachlicher Sicht, oder verschiedenen Aggregationsleveln.
- Views können auch gezielt dazu eingesetzt werden, um die Performance von SAS-Skripten zu verbessern. Es können viele langsame Eingabe-Ausgabe-Operationen eingespart werden, wenn die Quelldaten als View eingelesen werden. Bei der klassischen Verarbeitung per Tabelle werden die Daten erst eingelesen, transformiert und dann wieder auf Festplatte zurückgeschrieben und im nächsten Verarbeitungsschritt wieder von Festplatte eingelesen. Durch den zielgerichteten Einsatz einer View kann das Speichern und Lesen des Zwischenergebnisses auf Festplatte eingespart werden.
- Zuletzt können Views dazu eingesetzt werden, um komplizierte Datenabfragen (Joins, Unions, Filter, …) vor dem Endanwender zu verbergen. Der Anwender kann die Views z.B. direkt in seinen Berichten verwenden und muss sich nicht mit der Datenbereitstellung der Daten befassen.

Aber auch die Nachteile von Views sollen nicht verschwiegen werden:

- Im Vergleich zu der Verarbeitung von SAS Tabellen generieren Views immer einen kleinen Overhead.
- Je nach Programmablauf und den verwendeten SAS Prozeduren kann sich die Verwendung einer View auch negativ auf die Performance auswirken. Werden z.B. rechenintensive Views im gleichen Programm mehrfach verarbeitet, kann dies die Performance dramatisch verschlechtern.
- Wenn SAS auf die View nicht sequenziell, sondern indiziert zugreift, müssen erst alle Daten in ein sog. SPILL-File auf Festplatte materialisiert werden, was wiederum die Verarbeitungsgeschwindigkeit enorm ausbremst.

2.3 Programmierbeispiele

Es gibt zwei Arten, um eine View zu erzeugen: a) per klassischem DataStep und b) mit PROC SQL.
Die folgenden Beispiele zeigen, wie sich mit wenigen Zeilen Quellcode eine Sicht mit dem Namen „class_girls“ auf die SASHELP.CLASS Tabelle erzeugen lässt, die lediglich die Mädchen aus der Klasse enthält. Es wird auch gezeigt, wie man sich die Definition einer View und die darin gespeicherte Abfrage ins SAS Log ausgeben lassen kann.

Programmbeispiel für PROC SQL:

```
proc sql;
   /* view definieren */
   create view class_girls as
      select *
      from sashelp.class
      where sex = "F";

   /* Definition auslesen */
   describe view class_girls;
quit;
```

Programmbeispiel per DataStep:

```
/* view definieren */
data class_girls /view=class_girls;
   set sashelp.class(where=(sex="F"));
run;

/* Definition auslesen */
data view=class_girls;
   describe;
run;
```

3 Beispiele aus der Praxis

Wie sich aus den Beispielen erahnen lässt, sind SQL Views auf den Funktionsumfang von PROC SQL begrenzt, während DataStep Views über den vollen Funktionsumfang der DataStep Programmierung verfügen.
Hieraus ergeben sich interessante Einsatzmöglichkeiten und Lösungsansätze für einige praxisnahe Probleme.

3.1 Abfrage von Webservices

In vielen Unternehmen wächst zunehmend das Bedürfnis neben internen Datenquellen auch Daten von öffentlichen Diensten zu konsumieren und mit internen Daten zusammenzuführen. Eine Möglichkeit, um externe Daten anzubinden sind Webservices. Im einfachsten Fall handelt es sich hierbei um einen Standard URL-Aufruf, der die Daten bereits in strukturierter Form zurückliefert. Über Dienste wie Quandl lassen sich z.B.

über den Aufruf einer simplen URL[1] die Kursstände des Deutschen Aktienindex DAX als CSV Datei abfragen.

Mit Hilfe der SAS URL Filename Engine lassen sich die Daten direkt in einen DataStep einlesen. Somit kann man sämtliche Datenverarbeitungslogik innerhalb eines DataSteps abbilden und daher auch als View bereitstellen:

```
data DAX_Kurse / VIEW=DAX_Kurse;
  INFILE "https://www.quandl.com/api/v1/datasets/YAHOO/INDEX_GDAXI
.csv" URL (…);
  INPUT (…)
run;
```

Hinweis: die Beispiele in diesem Unterkapitel werden in abstrakter Form präsentiert, um die dahinterstehenden Konzepte und Anwendungsfälle zu vermitteln. Den vollständigen Quellcode finden sie im Anhang.

Bei jeder Abfrage der View wird nun im Hintergrund der Webservice aufgerufen und der Konsument der View erhält immer die aktuellsten Daten und muss sich nicht mit der Datenbereitstellung befassen.

Dank neuer Funktionalität seit SAS 9.3 lassen sich mit Hilfe von Seiteneffekten und der DOSUBL-Funktion sogar wesentlich komplexere Webserviceabfragen in eine View verpacken, wie im nächsten Beispiel per PROC HTTP.
Der Trick besteht darin, zu Beginn per DOSUBL-Funktion SAS Code in einem separaten Thread auszuführen, der die komplexe Datenabfrage und -aufbereitung ausführt und als Zwischenergebnis im Dateisystem ablegt.

Im zweiten Schritt wird diese dynamisch generierte Datei dann wieder eingelesen. Da die Datei zu Beginn des DataSteps aber noch nicht existiert, wird die DUMMY Filename Engine verwendet und erst zur Laufzeit wird über die Variable „fullpath“ der Ablageort der Datei dem INFILE Statement bekannt gegeben:

```
data DAX_Kurse2 / VIEW=DAX_Kurse2;
  IF (_n_ = 1) THEN DO;
    rc = dosubl('filename webfile "C:\temp\INDEX_GDAXI.csv";
proc http
url="https://www.quandl.com/api/v1/datasets/YAHOO/INDEX_GDAXI.csv"
out=webfile METHOD="GET"; run;');
  END;

  fullpath = "C:\temp\INDEX_GDAXI.csv";
  INFILE DUMMY FILEVAR=fullpath (…);

  INPUT (…);
run;
```

[1] https://www.quandl.com/api/v1/datasets/YAHOO/INDEX_GDAXI.csv

3.2 Dictionary Views

Erfahrene SAS Entwickler kennen und lieben die Vorzüge der von SAS bereitgestellten sogenannten Dictionary Tabellen. Sie sind in der SASHELP-Bibliothek leicht an dem Präfix „V“ zu erkennen, z.B. enthält die Dictionary Tabelle SASHELP.VTABLE (bzw. innerhalb von PROC SQL auch als Tabelle DICTIONARY.TABLES bekannt) Detailinformationen zu allen Tabellen aller Bibliotheken in der aktiven SAS Sitzung. Der Programmierer kann über diese Dictionary Tabellen z.B. sehr einfach datengetriebene Programme schreiben.

Bei der Suche werden allerdings Tabellen, die in den SAS Metadaten registriert sind, deren Bibliothek jedoch nicht allokiert ist, nicht gefunden. Als Abhilfe bietet es sich daher an, die relativ komplizierte Abfrage der SAS Metadaten in eigene Views zu kapseln und wiederverwendbar einer Vielzahl von Anwendern zur Verfügung zu stellen. Das folgende Beispiel fragt alle SAS Metadatentabellen ab, für die der Benutzer berechtigt ist und ermittelt den Tabellennamen (sowohl physisch als auch metadatenseitig), sowie die Metadaten-ID des Tabellenobjekts:

```
data metadata_tables /view=metadata_tables;
  query = "omsobj:PhysicalTable?@Id contains '.'";
  length TableUri id name tablename $256;

  nobj=metadata_getnobj(query,1,TableUri);

  do i=1 to nobj;
     nobj=metadata_getnobj(query,i,TableUri);

     /* Get Table attributes */
     rc=metadata_getattr(TableUri,'id',id);
     rc=metadata_getattr(TableUri,'Name',name);
     rc=metadata_getattr(TableUri,'TableName',tablename);

     output;
  end;
run;
```

Über weitere Abfragen lassen sich beispielsweise auch noch Informationen zum Metadatenordner hinzufügen.
Sämtliche Metadaten, die dem Benutzer zugänglich sind, können auf diese Weise abgefragt werden und als Dictionary View aufbereitet werden: Benutzer, Gruppen, Stored Processes, DI-Jobs, Spaltendefinitionen von Tabellen, usw..

3.3 Smart Prompts

Im Umfeld der SAS Plattform kommt häufig das SAS Prompting Framework zum Einsatz. Über vordefinierte Eingabemasken können insbesondere Endanwender über eine grafische Oberfläche die Ausführung eines Stored Processes oder Enterprise Guide

Projekts parametrisieren und somit typische Fehlerquellen, wie beispielsweise Tippfehler vermeiden.
Leider bietet das Prompting Framework von Hause aus keine Möglichkeit, um Abfrageprompts mit einer Berechnungslogik zu hinterlegen. Man kann nur statische Listen oder die Inhalte einer in den Metadaten registrierten Datenquelle angeben.
Letzteres bietet nun die Möglichkeit eine DataStep View mit einer benutzerdefinierten Verarbeitungslogik als Datenquelle zu verwenden. Bei jeder Abfrage des Prompts wird daher die View und damit die darin enthaltene Logik ausgeführt. Somit wird aus einer statischen Datenquelle eine „smarte“ View mit dynamischen Inhalten.

In der Praxis hat sich diese Vorgehensweise bereits mehrfach bewährt, z.B. um beim Starten eines STPs über den Browser (Stored Process Web Application) dem Anwender und dem Entwickler unterschiedliche Auswahlmöglichkeiten zu bieten. Die View prüft bei ihrer Ausführung, ob sie auf einem Entwicklungs- oder Testrechner ausgeführt wird und bietet dem Entwickler dann zusätzliche Testumgebungen, Testdaten oder Debugging-Einstellungen an. Auf der Produktivumgebung wird dieser Codepfad jedoch niemals durchlaufen und daher bleiben dem Anwender die erweiterten Testeinstellungen verborgen.
Der Stored Process kann somit jederzeit gefahrlos zusammen mit der View paketiert und ausgerollt werden, ohne Gefahr zu laufen, umgebungsspezifische Einstellungen zu überschreiben. Wer in der Produktion schon einmal eine Konfigurationstabelle mit den Daten aus der Testumgebung überschrieben hat, kennt vielleicht das Problem.

Da diese Verarbeitungslogik relativ komplex werden kann, wird hier ein etwas einfacheres Beispiel vorgestellt. Die Beispiel-View liefert zur Ausführungszeit über normale DataStep Funktionen eine Liste aller Dateien, die im Dateisystemordner „C:\KSFE_2016“ enthalten sind:

```
data PermLib.Prompt / VIEW= PermLib.Prompt;
   length folder name $256;

   folder = "C:\KSFE_2016";

   rc=FILENAME('mydir', folder);
   dirid=DOPEN('mydir');
   numfiles=dnum(dirid);
   do i=1 to numfiles;
      name=dread(dirid,i);
      output;
   end;

   rc=DCLOSE(dirid);
   rc=FILENAME('mydir');
   keep folder name;
run;
```

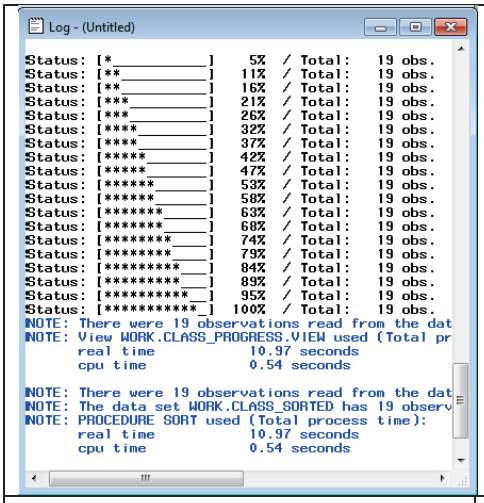
Sobald diese View nun in den Metadaten registriert wurde, kann sie als Datenquelle für das Prompting-Framework herangezogen werden und liefert nun eine dynamische Sicht auf das Dateisystem.

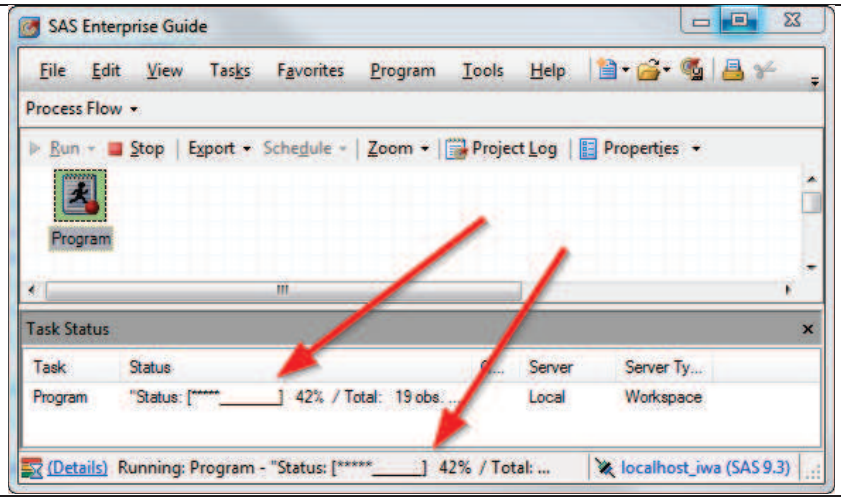
3.4 Fortschrittsmeldungen

Gerade bei größeren Datenmengen und lange dauernden Operationen, fragt man sich als Entwickler so manches Mal, wie weit die Verarbeitung wohl schon fortgeschritten ist und ob es sich lohnt, noch etwas länger zu warten, oder den Verarbeitungsjob gleich ganz abzubrechen.
Auch hierzu kann man mit Hilfe einer einfachen DataStep View eine schnelle Antwort bekommen. Der Lösungsansatz besteht darin, sich um die Datenquelle herum eine View mit der gleichen Datenstruktur zu bauen und innerhalb der View Fortschrittsmeldungen zu generieren.
Arbeitet man interaktiv im SAS Display Manager, kann man mit einem PUTLOG Statement direkt eine Nachricht in das SAS Log schreiben. Bei der moderneren Variante mit dem SAS Enterprise Guide kann man mit dem SYSECHO Statement eine Nachricht an den Client schicken. Anhand der Statusmeldungen kann man nun sofort sehen, wie weit die Verarbeitung z.B. innerhalb eines PROC SORT fortgeschritten ist.

```
data class_progress /view=class_progress;
   length message $200 nobs 8;
   set sashelp.class nobs=nobs;

   /* Nachricht zusammensetzen */
   message = cat("Status: [",
          repeat("*",floor(_N_ /nobs*10)),
          repeat("_",10-floor(_N_ /nobs*10)),
          "] ",
          put(_N_/nobs, percent.),
          " / Total: ",
          put(nobs,comma4.),
          " obs.");

   /* Nachricht senden */
   rc = dosubl(cat('SYSECHO "', message, '";'));
   putlog message;

   /* große Datenmenge simulieren */
   call sleep(500, .001);

   drop nobs message;
run;

/* View verarbeiten */
proc sort data=class_progress out=class_sorted;
   by name;
run;
```

Die Statusmeldungen präsentieren sich dann während der PROC SORT Ausführung wie folgt:





Log im Display Manager	Enterprise Guide Ausgabe in Statusleiste und Fenster Anwendungsroutinenstatus

3.5 Row-Level Security

Zuletzt spielen Views eine tragende Rolle bei der Implementierung einer Row-Level Security Strategie. Hierbei bekommt der Endanwender zwar Zugriffsrechte auf die View, ist jedoch nicht berechtigt alle Datensätze einer Tabelle zu lesen. Je nach angemeldetem Benutzer und seinen Gruppenzugehörigkeiten darf der Anwender dann nur noch bestimmte Zeilen lesen, die z.B. zu seinem Vertriebsgebiet oder seiner Abteilung zugeordnet sind.

Ab SAS Version 9.3 werden sog. Metadata-Bound Libraries (auch: Secured Libraries) unterstützt, die es ermöglichen, bei der Abfrage einer vordefinierten SQL View auf die Identität des angemeldeten Metadatenbenutzers und seiner Gruppenberechtigungen zuzugreifen und auf diesem Wege per WHERE Filter diejenigen Datensätze auszublenden, für die der Anwender nicht berechtigt ist.

Aus Platzgründen wird in diesem Beitrag auf ein Implementierungsbeispiel verzichtet. Die SAS Dokumentation „Guide to Metadata-Bound Libraries“[2] bietet eine gute Übersicht und Einleitung in das Thema inkl. Codebeispiel. Unter SAS 9.4 wurden die Möglichkeiten zudem noch einmal erweitert und man kann nun z.B. auch über das Betriebssystembenutzerkonto oder die Zugehörigkeit zu Metadaten-Benutzergruppen filtern[3].

[2] http://support.sas.com/documentation/cdl/en/seclibag/66930/HTML/default/viewer.htm#p0t0jlaj40gwt7n1tvvv43fe8nby.htm
[3] http://support.sas.com/documentation/cdl/en/bisecag/67045/HTML/default/viewer.htm#n0m3ptctbmjpzkn1woe49pep6tep.htm

4 Fazit und Ausblick

Die vorgestellten Praxisbeispiele konnten dem Leser hoffentlich einen Eindruck vermitteln, wie mächtig selbst definierte Views werden können, wenn man sie richtig einsetzt. Wiederkehrende Verarbeitungsschritte und deren Ablauflogik lassen sich nicht nur als Quellcode bzw. Makro kapseln, sondern auch fix und fertig als wiederverwendbare Datenansicht bereitstellen.
Diese Views lassen sich nun im nächsten Verarbeitungsschritt bequem in Berichten oder Abfrage-Prompts verwenden, oder in interaktiven Analysetools wie Enterprise Guide öffnen und weiterverarbeiten.

Da die komplette Funktionalität der DataStep Programmierung in einer DataStep View ausreizt werden kann, stehen dem Programmierer nahezu unbegrenzte Möglichkeiten offen. Der Beitrag konnte den Leser hoffentlich dazu anregen eine neue Ansicht von SAS Sichten zu gewinnen und zu der Einsicht zu kommen, dass sich viele Probleme sehr elegant mit einer View lösen lassen. In diesem Sinne wünscht der Autor dem Leser viel Spaß beim Experimentieren mit eigenen Problemstellungen.

Anhang A: Codebeispiele

Beispiele aus Kapitel 3.1 Webservices:

```
data DAX_Kurse / VIEW=DAX_Kurse;
  INFILE "https://www.quandl.com/api/v1/datasets/YAHOO/INDEX_GDAXI.csv"
         URL
         FIRSTOBS=2
         ENCODING="WLATIN1"
         TERMSTR=LF
         DLM=','
         MISSOVER
         DSD;

  FORMAT datum yymmdd10.;
  INPUT
         datum             : ?? YYMMDD10.
         Open              : ?? COMMA8.
         High              : ?? COMMA8.
         Low               : ?? COMMA8.
         Close             : ?? COMMA8.
         Volume            : ?? COMMA11.
         Adjusted_Close    : ?? COMMA8. ;
run;
```

```
data DAX_Kurse2 / VIEW=DAX_Kurse2;
  IF (_n_ = 1) THEN DO;
    rc = dosubl('filename webfile "C:\temp\INDEX_GDAXI.csv"; proc http
url="https://www.quandl.com/api/v1/datasets/YAHOO/INDEX_GDAXI.csv"
out=webfile METHOD="GET"; run;');
    DROP rc;
  END;

  fullpath = "C:\temp\INDEX_GDAXI.csv";
  INFILE DUMMY
        FIRSTOBS=2
        ENCODING="WLATIN1"
        TERMSTR=LF
        DLM=','
        MISSOVER
        DSD
        END=LAST
        FILEVAR=fullpath;

  FORMAT datum yymmdd10.;
  INPUT
        datum            : ?? YYMMDD10.
        Open             : ?? COMMA8.
        High             : ?? COMMA8.
        Low              : ?? COMMA8.
        Close            : ?? COMMA8.
        Volume           : ?? COMMA11.
        Adjusted_Close   : ?? COMMA8. ;

run;
```