Datenmanagement mit Oracle, SAS, Perl und Unix-Utilities: Werkzeuge für alle Fälle

Rainer Kaluscha
Forschungsinstitut für Rehabilitationsmedizin
an der Universität Ulm
Karl-Wilhelm-Heck-Str. 6
D-88410 Bad Wurzach
rainer.kaluscha@uni-ulm.de

Zusammenfassung

Bei der Auswertung von umfangreichen Datenbeständen ergeben sich vielfältige (Teil-) Probleme. Der Versuch, sie alle mit einem Software-Werkzeug zu lösen, scheitert oft oder führt zu unnötig umständlichen Lösungen. Durch geschicktes Kombinieren der Stärken verschiedener Tools lässt sich mühsame und fehlerträchtige Handarbeit sparen.

Schlüsselwörter: Datenmanagement, Utilities, Tools, Perl, Datenbank

1 Einleitung

Während sich kleinere Projekte noch von der Datenerfassung bis zur Datenauswertung komplett in SAS abwickeln lassen, ist dies bei großen Datenbeständen und umfangreichen Auswertungen kaum mehr praktikabel. Hier bieten dann Datenbankmanagementsysteme[1] bei der Datenhaltung und -aufbereitung sowie Perl-Skripte und die Standard-Unix-Utilities zur Weiterverarbeitung des mitunter recht umfangreiches SAS-Outputs, der leicht mehrere hundert Druckseiten erreichen kann, Unterstützung. Mancher wird aus seinen .log-Dateien Meldungen wie diese kennen:

```
NOTE: The PROCEDURE MEANS printed pages 363-365.
```

Das manuelle Durcharbeiten wird dann mühsam und zeitaufwändig, so dass Computerunterstützung bei der Extraktion der relevanten Informationen mehr als willkommen ist. Einige Möglichkeiten sollen im folgenden am Beispiel des Projektes „Reha-Net“ [1] dargestellt werden. Dort werden in anonymisierter Form Routinedaten aus der Rehabilitation ausgewertet. Der Datenbestand umfasst derzeit mehr als 100.000 Datensätze mit gut 500 Variablen, darunter auch etliche Freitextfelder. Nicht nur die reine Menge, sondern auch der dynamische Zufluss und die Struktur der Daten erfordern den Einsatz eines leistungsfähigen Datenbankmanagementsystems wie in diesem Falle Oracle.

[1] Tabellenkalkulationen wie z.B. MS Excel gehören übrigens nicht dazu; auch MS Access fehlen wesentliche Elemente eines Datenbankmanagementsystems wie z.B. Views oder Transaktionen. Hier ist an professionelle Datenbankmanagementsysteme wie Oracle, DB2, MySQL oder MS SQLServer gedacht.

2 Datenhaltung

Datenbankmanagementsysteme bieten effiziente Möglichkeiten zur Speicherung, zum Wiederauffinden und Verknüpfen großer Mengen komplexer Informationen. Darüber hinaus erlauben sie auch die Prüfung von Plausibilitätskriterien und Integritätsbedingungen, verwalten den konkurrierenden Zugriff mehrerer Benutzer, prüfen Zugangsberechtigungen und bieten Mechanismen zum Schutz vor Datenverlust [2].

Zurück zu unserem Beispiel: so kann es etwa zu jedem Rehabilitanden beliebig viele verschiedene Laborwerte geben. Zu jedem Laborwert können wiederum beliebig viele Wiederholungsmessungen vorliegen. Solche Strukturen sind einerseits in einem SAS-Table nicht abzubilden; andererseits aber für eine statistische Auswertung auch nicht geeignet. Hier bedarf es einer zweigleisigen Lösung: Die Speicherung erfolgt in der Datenbank in mehreren über 1:n-Beziehungen verknüpften Tabellen. Für die Auswertungen werden über entsprechende Views (Datenbanksichten) die jeweils benötigten Daten bereitgestellt.

Geht es bei einer Auswertung etwa um kardiovaskuläre Risiken, werden nicht sämtliche Laborwerte benötigt, sondern insbesondere die Blutfette (Gesamtcholesterin, HDL, LDL, Triglyceride, etc.). Von diesen wiederum werden auch nicht alle Einzelwerte benötigt, sondern man wird sich aus Gründen der Standardisierung auf die Werte zu einem bestimmten Zeitpunkt (z.B. Beginn der Behandlung) oder einen Durchschnittswert beschränken. Sind die entsprechenden Festlegungen erfolgt, lässt sich mit wenigen Zeilen SQL der entsprechende View erzeugen:

```
create view v_herzinfarktrisiko as
select p.pat_id, p.alter, p.geschlecht,
       avg( decode( l.code, 'HDL+', l.wert, NULL ) ) hdl,
       avg( decode( l.code, 'LDL+', l.wert, NULL ) ) ldl
from patienten p, laborwerte l
where p.pat_id = l.pat_id
  and l.code in ( 'HDL+', 'LDL+' )
group by p.pat_id, p.alter, p.geschlecht;
```

Dieser View verknüpft die Tabelle `patienten` über die Patienten-ID mit der Tabelle `laborwerte` und stellt für alle Patienten, bei denen entsprechende Laborwerte vorliegen, die Patienten-ID, Alter und Geschlecht sowie die durchschnittlichen HDL- und LDL-Werte bereit. Dabei kann das Datenbankmanagementsystem seine Stärken ausspielen, z.B. das effiziente Suchen und Verknüpfen von Daten. So verfügen Datenbankmanagementsysteme über ausgefeilte Techniken zur Anfrageoptimierung (*query optimizer*): Statt für alle Patienten nachzuschauen, ob entsprechende Laborwerte vorhanden sind, kann es effizienter sein, insbesondere wenn ein Index auf dem Feld `code` der Tabelle `laborwerte` angelegt wurde und viele verschiedene Arten von Laborwerten vorliegen, zunächst die passenden Laborwerte herauszusuchen und anschließend über die Patienten-ID, auf der als Primärschlüssel ebenfalls ein Index liegt, Alter und

Geschlecht aus der Tabelle `patienten` zu ergänzen. So kann das Datenbankmanagementsystem sehr schnell die angeforderten Daten aus Millionen unterschiedlicher Datensätze bereitstellen.

Moderne Datenbankmanagementsysteme bringen auch grafische Oberflächen zur Erzeugung solcher Views mit. Diese bieten aber nur eingeschränkte Möglichkeiten; der volle Funktionsumfang erschließt sich – ähnlich wie bei SAS und Analyst – erst über die SQL-Syntax. Kleine Tricks wie das Ausblenden der jeweils anderen Laborwerte über `decode` bei der Berechnung der Durchschnittswerte sind dort nicht vorgesehen. Wird ein View über solche Oberflächen „zusammengeklickt", sollte auch daran gedacht werden, diesen zur Dokumentation zu speichern, damit später nachvollzogen werden kann, auf welchem Weg die Daten selektiert wurden.

Über die SAS-Access-Module kann auf Datenbanktabellen und -views praktisch wie auf SAS-Data-Sets zugegriffen werden (Views sind allerdings nicht beschreibbar). Da bei unserem Projekt auch laufend neue Daten aus dem Routinebetrieb nachströmen, empfiehlt es sich, von dem View einen „Schnappschuss" in Gestalt eines SAS-Data-Sets zu erstellen, um während der Auswertung eine zeitlich stabile Basis zu haben.

```
/* Oracle-DB als SAS-Lib deklarieren */
libname rehanet oracle user=ich password=geheim path=dbserver;
data sasuser.herzinfarktrisiko;
    set rehanet.v_herzinfarktrisiko;
```

Der Datenimport über einen Datenbankview bietet übrigens gegenüber dem Import aus einer Datei den Vorteil, dass SAS die Datentypen der Variablen automatisch bestimmen kann.

Im Projekt „Reha-Net" werden auch aus Freitexten über die von Oracle angebotenen computerlinguistische Komponente und entsprechende Views Informationen extrahiert und zur Auswertung nach SAS übergeben [3]. So kann z.B. das Vorhandensein bzw. Fehlen von textuellen Hinweisen in der Genussmittelanamnese darauf, dass der Patient raucht, als binäre Variable „Raucher ja / nein" in einem Datenbankview erscheinen.

3 Datenauswertung

Nach dem Import der Daten kann man nun die Stärken von SAS bei komplexen statistischen Auswertungen nutzen. Aber auch hier kann man sich das Leben durch die Nutzung weiterer Software [4] mitunter etwas leichter machen. So sind oft ähnliche Auswertungsschritte für viele Variable zu wiederholen.
Dann kann man sich für die nötigen SAS-Kommandos eine Schablone erstellen bzw. von SAS-Analyst erstellen lassen. Anschließend generiert man mit Perl- oder Standard-Unix-Utilities, insbesondere Shell-Skripten, daraus SAS-Code:

```
for i in ges 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
do
    cat <<- EOF
    bogen_${i}_d31       = bogen_${i}3   - bogen_${i}1;
    bogen_${i}_d21       = bogen_${i}2   - bogen_${i}1;
    bogen_${i}_d32       = bogen_${i}3   - bogen_${i}2;
EOF
done

for i in ges 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
do
        echo "  bogen_${i}_d31  bogen_${i}_d21  bogen_${i}_d32"
done
```

Dieses kleine Shell-Script erzeugt in der ersten for-Schleife den nötigen SAS-Code, um im Data-Step für den Gesamtscore und die 19 Items eines Fragebogens (VAS-Wirbelsäulenscore) die Differenzen zwischen der ersten, zweiten und dritten Nachuntersuchung zu berechnen. Der Ausdruck `${i}` in der Schablone wird dabei sukzessive durch alle Variablennamen aus der Liste ersetzt.
Die zweite for-Schleife erzeugt nochmal die Variablennamen für den VAR-Abschnitt der `PROC Univariate.`
Die Ausgabe des Shell-Scripts wird dann einfach in eine Datei umgelenkt und manuell die fehlenden Zeilen für den Data-Step und den Aufruf der Prozedur ergänzt:

```
data work.tmp0;
    set sasuser.rueckenschmerz;

    bogen_ges_d31 = bogen_ges3 - bogen_ges1;
    bogen_ges_d21 = bogen_ges2 - bogen_ges1;
    bogen_ges_d32 = bogen_ges3 - bogen_ges2;

    bogen_1_d31   = bogen_13   - bogen_11;
    bogen_1_d21   = bogen_12   - bogen_11;
    bogen_1_d32   = bogen_13   - bogen_12;

    ...


proc univariate data=work.tmp0;
  var   bogen_ges_d31   bogen_ges_d21   bogen_ges_d32
        bogen_1_d31     bogen_1_d21     bogen_1_d32
        ...
```

So kann mit wenig Aufwand getestet werden, ob zwischen den Untersuchungszeitpunkten bei einer der 20 Variablen signifikante Veränderungen aufgetreten sind. Nun gilt es nur noch, aus dem „geschwätzigen“ Output von `PROC Univariate` die Werte des Vorzeichenrangtests (signed rank test) auszufiltern.

4 Aufbereitung von SAS-Output

Für diese Aufgabe ist Perl (Practical Extraction and Report Language) [5] das Werkzeug der Wahl. Die Programmiersprache wurde entwickelt, um Textdateien effizient zu verarbeiten und bietet mit den regulären Ausdrücken (regular expressions) [6] mächtige Such- und Ersetzungsfunktionen. Die Syntax wirkt zwar auf den ersten Blick etwas kryptisch, lässt sich aber dennoch überraschend leicht erlernen.

Der SAS-Output ist recht regelmäßig strukturiert und lässt sich daher gut verarbeiten. Folgendes kleines Perl-Skript extrahiert aus den immerhin 4824 Zeilen, die vorstehender Aufruf der `PROC Univariate` erzeugte, die 60 Zeilen mit den Werten des Vorzeichenrangtests:

```
 1  #! /usr/bin/perl -w
 2
 3  my $var;
 4  my $p;
 5
 6  while (<STDIN>)
 7  {
 8          # get variable
 9          if( /\s*Variable:/ )
10          {
11                  ($var) = m{\s*Variable:\s+(\w+)};
12          }
13
14          # get p value
15          if( /\s*Signed Rank/ )
16          {
17                  ($p) = m{\|S\|\s+([\d\.< ]+)};
18                  printf "%-20s %7s\n", $var, $p;
19          }
20  }
```

Die Schleife in Zeile 6 liest zeilenweise von der Standardeingabe, die von der .lst-Datei gespeist wird. In Zeile 9 werden Zeilen mit einem Variablennamen erkannt und dieser wird ggf. anhand des regulären Ausdrucks `\s*Variable:\s+(\w+)` in Zeile 11 extrahiert. Reguläre Ausdrücke beschreiben Zeichenketten nach formalen Kriterien. Der Ausdruck in Zeile 11 sucht nach beliebig vielen Leerzeichen (`\s*`), gefolgt von dem

Text „`Variable:`“, mindestens einem weiteren Leerzeichen (`\s+`) und einer Zeichenfolge aus Buchstaben und Ziffern mit mindestens einem Zeichen (`(\w+)`). Diese Zeichenfolge ist der gesuchte Variablenname; die runden Klammern um den Teilausdruck wählen ihn für die Zuweisung an die Variable `$var` aus.
Wird nun in der Folge eine Zeile mit dem Schlüsselwort `Signed Rank` gefunden (Zeile 15), wird analog die hinter dem Text `Pr >= |S|` stehende Zahl als gesuchter p-Wert extrahiert (Zeile 17) und beide ausgegeben (Zeile 18).
Leitet man die Ausgabe des Perl-Skriptes noch durch das Sortierkommando (`sort -n -k 2`), erhält man die kleinsten p-Werte zuerst und somit schnell einen Überblick:

```
...
bogen_18_d31          0.0098
bogen_12_d32          0.0173
bogen_9_d21           0.0173
bogen_18_d32          0.0298
bogen_6_d31           0.0394
bogen_ges_d21         0.0438
bogen_3_d32           0.0465
bogen_10_d31          0.1119
bogen_16_d21          0.1233
...
```

Dieses Vorgehen erleichtert nicht nur die Arbeit, sondern beugt auch dem Übersehen wichtiger Ergebnisse vor, indem der umfangreiche SAS-Output maschinell auf das für die Fragestellung wesentliche reduziert wird.

5 Diskussion

Zwar bedeutet die Einarbeitung in neue Software einen gewissen Aufwand. Dies gilt besonders für Datenbanksysteme, falls nicht schon ohnehin eine Studiendatenbank existiert.
Dieser Aufwand amortisiert sich aber recht bald wieder, zumal viele Problemstellungen immer wieder in ähnlicher Form auftauchen und einmal erstellte Skripte – ggf. mit kleinen Änderungen – immer wieder verwendet werden können. Wenn man mit einem Tool noch nicht vertraut ist, finden sich vielleicht auch im Kollegen- oder Bekanntenkreis bzw. im Internet Vorlagen, die man an seine Bedürfnisse anpassen kann.

Literatur

[1] Kaluscha R, Jacobi E: *Eine Datenbank zur Effektivitätsbeurteilung: Das Datenkonzept des rehabilitationswissenschaftlichen Forschungsverbundes Ulm.* DRV-Schriften 20, S. 218-219 (2000).

[2] Elmasri R & Navathe SB: *Grundlagen von Datenbanksystemen*, Ausgabe Grundstudium, 3. Auflage. Pearson Studium, Addison-Wesley Longman, Amsterdam (2005). vgl. auch: Wikipedia: *Datenbanksystem.* Online: http://de.wikipedia.org/wiki/Datenbanksystem (13.02.2007)

[3] Kaluscha R: *Informationsgewinnung aus Freitexten in der Rehabilitationsmedizin.* Dissertation, Universität Ulm, Medizinische Fakultät (2005). Online: http://vts.uni-ulm.de/doc.asp?id=5265

[4] Unter Unix gehören diese Utilities zum Standard, für Windows sind sie als GNU Utilities for Win32 verfügbar: Online: http://sourceforge.net/projects/unxutils (13.02.2007)

[5] *Perl Directory*. Online: http://www.perl.org (13.02.2007)

[6] Wikipedia: *Regulärer Ausdruck*. Online: http://de.wikipedia.org/wiki/Regul%C3%A4rer_Ausdruck (13.02.2007)