Umkodieren von ICD-9-Daten zu ICD-10 in SAS mittels einer relationalen Datenbank und PROC SQL

Andreas Deckert
Institute of Public Health
INF 324
69120 Heidelberg
a.deckert@uni-heidelberg.de

Zusammenfassung

Mit SQL existiert eine mächtige Prozedur innerhalb von SAS, mit der die volle Funktionsfähigkeit der Datenbanksprache SQL (Structured Query Language) für die Abfrage und Verknüpfung von Daten und Tabellen zur Verfügung steht. Oftmals wird PROC SQL benutzt, um z.B. innerhalb eines Makros Einträge in einer Tabelle zu zählen und diese dann an eine globale Makrovariable zu übergeben oder um komplexe Verknüpfungen von Tabellen und Einträgen vorzunehmen. Es ist jedoch darüber hinaus auch möglich, mit PROC SQL eine komplette relationale Datenbank innerhalb von SAS aufzubauen. Damit ist die Handhabung von Wertebereichsintegritäten und referentiellen Integritäten innerhalb von SAS möglich, so dass auf externe Datenbanken verzichtet werden kann. Das Prinzip des Aufbaus einer relationalen Datenbank wird hier anhand eines einfachen Beispiels vorgestellt: Für die Analyse von Todesursachen über einen längeren Zeitraum hinweg stehen in der Regel ICD[1]-codierte Daten zur Verfügung. Überdeckt die Analyse einen längeren Zeitraum, liegen die Daten allerdings bis 1998 nach ICD Version 9 und danach nach ICD Version 10 kodiert vor. Für eine einheitliche Auswertung nach ICD10 stellt das DIMDI[2] eine sogenannte Umsteigertabelle bereit. Diese Umsteigertabelle wird hier in eine relationale Datenbank innerhalb SAS überführt und dann für die Umcodierung von ICD9-Daten nach ICD10 verwendet.

Schlüsselwörter: PROC SQL, relationale Datenbank, ICD9, ICD10, Umkodierung

1 Einleitung: relationale Datenbanken mit PROC SQL

Die Prozedur SQL bietet die Möglichkeit, innerhalb von SAS die komplette Syntax der standardisierten Datenbanksprache SQL zu verwenden. SQL wurde für die Handhabung und Abfrage relationaler Datenbanken entwickelt, wofür vor allem die Beziehungen zwischen einzelnen Tabellen von Bedeutung sind. Das Aufsplitten von einzelnen komplexen Daten-Objekten in viele verschiedene durch Relationen verbundene Tabellen bietet z.B. den Vorteil, mehrdimensionale Einträge einfach zu handhaben, wodurch allerdings die Übersichtlichkeit über ein komplettes Datenobjekt verlorengeht. Grund-

[1] ICD: International Statistical Classification of Diseases and Related Health Problems; s. auch http://de.wikipedia.org/wiki/Internationale_statistische_Klassifikation_der_Krankheiten_und_verwandter_Gesundheitsprobleme
[2] DIMDI: Deutsches Institut für Medizinische Dokumentation und Information

Grundsätzlich können mit PROC SQL sämtliche Data Step Funktionalitäten sowie einige grundlegende SAS Prozeduren verwirklicht werden. PROC SQL bietet sich daher an, wenn es um die Verknüpfung von Data Sets (das Äquivalent zu Tabellen in SQL) für Abfragen/Manipulationen geht, wogegen SAS Data Steps vor allem für die Datenmanipulation innerhalb eines Data Sets geeignet sind. Im Vergleich zu Data Steps ist PROC SQL bei der Manipulation vieler kleinerer Data Sets deutlich schneller und benötigt zudem keine vorangestellten Sortier-Prozeduren. Generell können auch innerhalb eines Data Steps, z.B. mit MERGE oder SET mehrere Data Sets auf verschiedenste Weise zusammengeführt oder verknüpft werden. Der funktional äquivalente SQL-Code ist in der Regel allerdings wesentlich kompakter als die entsprechende Data Step Syntax. Ein paar der wichtigsten Verknüpfungen sind in der folgenden Abbildung schematisch dargestellt:

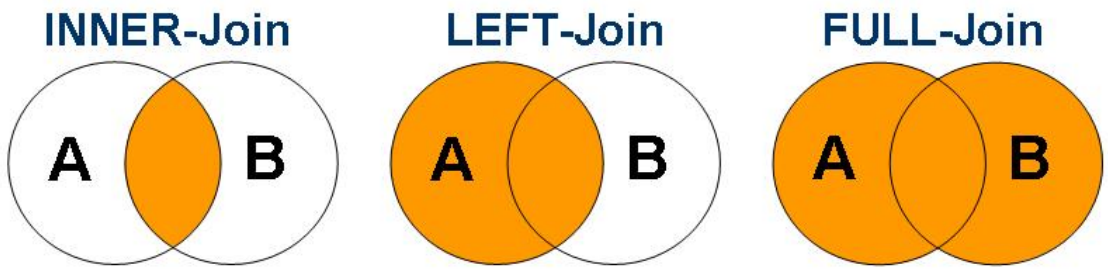

Abbildung 1: Tabellen-Verknüpfungen

Bei einer Verknüpfung zweier Data-Sets mittels INNER-Join werden zu jedem Eintrag aus Data Set A entsprechend der Verknüpfungsregel die passenden Einträge in Tabelle B gesucht und umgekehrt (kartesisches Produkt). Einträge aus A, zu denen es keine Entsprechung in B gibt und umgekehrt, werden nicht in die Ergebnis-Tabelle übernommen. Beim LEFT-Join dagegen werden auch Einträge aus dem Data Set A ohne Entsprechung in B übernommen, beim FULL-Join zusätzlich Einträge aus B ohne Entsprechung in A. Ein einfaches Beispiel einer Verknüpfung zweier Data Sets mit PROC SQL ist in den folgenden Code-Zeilen dargestellt:

```
PROC SQL;
     CREATE TABLE Ergebnis AS SELECT
         A.Medikament AS KH_Med, B.Medikament AS Kasse_Med,
         A.Diagnose, B.*
         FROM Krankenhaus A INNER JOIN Krankenkasse B
         ON A.Patient_ID=B.Patient_ID;
QUIT;
```

Hierbei entsteht eine Tabelle in der zu sämtlichen Generika die in einem Krankenhaus verordnet wurden, die entsprechenden Krankenkassen mit denen für diese Medikamente Rabattverträge bestehen aufgelistet werden (das können auch mehrere pro Generika sein). Im umgekehrten Fall werden zu jeder Krankenkasse diejenigen Medikamente aufgelistet, die dieses Krankenhaus mit dieser Krankenkasse abgerechnet hat.

Neben den Funktionen zur Verknüpfung von Tabellen kann PROC SQL auch benutzt werden, um z.B. arithmetische Operationen wie das Zählen aller Einträge vorzunehmen und das Ergebnis dann an eine Makro-Variable zu übergeben. Auch solche Operationen lassen sich mit PROC SQL einfacher umsetzen als mit Data Steps in Kombination mit der Übergabe von Parametern mittels CALL SYMPUT/ SYMGET.

Viele SAS-Nutzer setzen PROC SQL inzwischen für die oben beschriebenen Funktionalitäten routinemäßig ein[3] und sind mit den Operatoren von PROC SQL vertraut. Allerdings nutzen nur wenige das eigentlich ursprüngliche Potential der SQL-Datenbanksprache zum Aufbau einer kompletten relationalen Datenbank innerhalb von SAS. Dabei bietet die Verwendung einer relationalen Datenbank gerade innerhalb von SAS mehrere Vorteile: Zum einen entfallen dadurch zusätzliche externe Softwareprodukte wie z.B. Microsoft Access zur Verwaltung der Datenbank, zumal auch in SAS das Erstellen von Dateneingabemasken möglich ist [3]. Damit können von der Datenbankerstellung über die kontrollierte Dateneingabe bis zur Auswertung der Daten die kompletten Prozesse innerhalb von SAS umgesetzt werden. Zum anderen stehen die durch die relationale Datenbank festgelegten referentiellen Integritätsbedingungen sowie Wertebereichsintegritäten innerhalb von SAS zur Verfügung, was die Validität der Daten nach Verknüpfungen und Manipulationen gewährleistet. Es kann z.B. schon beim Erstellen der Datenbank festgelegt werden, dass bestimmte Variable keine fehlenden Werte enthalten dürfen oder nur bestimmte Werte innerhalb eines Bereiches einnehmen können. Verletzungen dieser Integritätsbedingungen bei Datenmanipulationen führen dann zu Fehlermeldungen, wogegen sie bei normalen Manipulationen ohne eine relationale Datenbank durch den Programmierer jedes Mal selbst geprüft werden müssen.

Ein weiterer großer Vorteil einer relationalen Datenbank ist die Verwendung von referentiellen Integritätsbedingungen zwischen einzelnen Tabellen bzw. Data Sets. Damit lassen sich Änderungen über mehrere Tabellen hinweg kontrollieren. So kann z.B. ein eindeutiger Primärschlüssel in einer Tabelle (z.B. eine ID) in einer anderen Tabelle als (redundanter) Fremdschlüssel verwendet werden. Durch die Referenzierung auf die Primärschlüssel-Tabelle können bei einer Änderung eines Eintrages in der Primärschlüssel-Tabelle notwendige Änderungen in der Fremdschlüssel-Tabelle mit durchgeführt werden. Durch referentielle Integritätsbedingungen sind vielfältige Beziehungen unter den Tabellen gestaltbar, wodurch Datenabfragen kontrollierbarer werden. Zudem sind Tabellen mit referentiellen Integritätsbedingungen vor unsachgemäßen Manipulationen und Löschen geschützt; dafür müssten zuerst die Integritätsbedingungen zwischen den Tabellen entfernt werden.

[3] Für einen einführenden Vergleich von PROC SQL und Data Steps siehe [1], eine umfangreiche Einführung bietet [2].

2 Problemstellung: Umkodieren von ICD9 zu ICD10

Für das hier verwendete Beispiel zur Demonstration des Aufbaus einer relationalen Datenbank in SAS ist eine relationale Datenbank nicht unbedingt erforderlich, es eignet sich aber gut um die Verwendung relationaler Datenbanken in SAS einführend zu erläutern.

In speziellen epidemiologischen Studien ist man z.B. an den Todesursachen innerhalb einer Risikogruppe interessiert. Hierfür werden die Todesursachen dieser Gruppe über einen längeren Zeitraum analysiert. Für kategorielle Analysen und Vergleichbarkeit mit anderen Studien werden die Todesursachen in der Regel nach dem WHO[4]-Klassifikationssystem ICD kodiert. Bei langen Beobachtungszeiträumen kann dabei der Fall auftreten, dass ein Teil der Daten in der älteren ICD9-Version und ein anderer Teil in der aktuellen ICD10-Version vorliegt. Um eine Analyse zu ermöglichen, müssen die Daten in eine einheitliche ICD-Version überführt werden. Bei der Einführung des ICD10 wurden allerdings umfangreiche Änderungen in der Struktur des Kodierungssystems vorgenommen, so wurde z.B. eine alpha-numerische Notation mit stärkerer Differenzierung eingeführt. Einige ICD-Gruppierungen aus ICD9 wurden gestrichen, neue wurden hinzugefügt und Gruppen-Einträge wurden in andere Gruppierungen verschoben. Damit ist eine einfache und eindeutige Überführung von ICD9-kodierten Daten in ICD10 nur in 60% der Fälle möglich.

Das Deutsche Institut für Medizinische Dokumentation und Information[5] (DIMDI) stellt auf seiner Homepage die kompletten ICD9- und ICD10-Codes sowie eine sogenannte Umsteigertabelle als Text-Files im ASCII-Format zum Download zur Verfügung [4]. Die beiden ICD-Tabellen enthalten neben den eindeutigen ICD-Codes der jeweiligen Version die Beschreibung des Krankheitsbildes und der Krankheitsursachen. Die Umsteigertabelle enthält zum einen zeilenweise die eindeutigen Umsetzungen von ICD9 zu ICD10 und umgekehrt, und zum anderen alle möglichen mehrdeutigen Umsetzungen. Es gibt also ICD9-Codes für die mehrere mögliche ICD10-Kodierungen existieren und ebenso ICD10-Codes die mehrere Entsprechungen in ICD9 haben. Andererseits gibt es aber auch mehrere ICD9-Codes für die derselbe ICD10-Code gültig ist und umgekehrt. Zusätzlich existiert eine Variable die im Falle von mehreren Zuordnungsmöglichkeiten von ICD10 zu ICD9 eine Zuordnung empfiehlt. Für die Umsetzung von ICD9 nach ICD10 existiert keine solche Variable, da die ICD10 stärker ausdifferenziert ist als die ICD9. Die Struktur der Text-Files ist in Abbildung 2 dargestellt.

Die Text-Files können nun zunächst in SAS importiert werden. Hierzu kann folgender Code benutzt werden (Beispiel für die ICD10-Tabelle):

```
DATA import_ICD10;
     %LET _EFIERR_ = 0;
```

[4] World Health Organization
[5] Homepage des DIMDI: www.dimdi.de

```
      INFILE "&Pfad.\ICD10.txt"
      DELIMITER = ';' MISSOVER DSD LRECL = 32767 FIRSTOBS = 1;
      FORMAT Code10 $7. Titel10 $255.;
      INPUT Code10 $ Titel10 $;
      IF _ERROR_ THEN CALL SYMPUTX('_EFIERR_',1);
RUN;
```

Dabei ist `&Pfad.` eine globale Variable, die den Pfad der Download-Datei auf der Festplatte enthält. Nachdem alle drei Dateien erfolgreich in SAS importiert worden sind, kann mit dem Aufbau der relationalen Datenbank begonnen werden.

- **ICD-9 Tabelle:**

```
410-414; Ischämische Herzkrankheiten
410;     Akuter Myokardinfarkt
...
414;     Sonstige Formen von chronischen ischämischen Herzkrankheiten
414.0;   Koronararteriosklerose
414.1;   Herzwandaneurysma
```

- **ICD-10 Tabelle:**

```
I21;     Akuter Myokardinfarkt
I21.0;   Akuter transmuraler Myokardinfarkt der Vorderwand
...
I22;     Rezidivierender Myokardinfarkt
I22.0;   Rezidivierender Myokardinfarkt der Vorderwand
```

- **Umsteiger-Tabelle:**

ICD-10	ICD-9	ICD-9(dreistellig)	1:1
I21.0;	410;	410;	A
I23.3;	429.8;	429;	A
I24.0;	410;	410;	

Abbildung 2: DIMDI Text-Files ICD9, ICD10 und Umsteigertabelle

3 Aufbau einer relationalen Datenbank in SAS

Die allgemeine Beschreibung des Aufbaus einer relationalen Datenbank mit diesen drei Tabellen ist in einem zusätzlichen Text-File im ZIP-Download der ICD-Tabellen des DIMDI enthalten und wurde hier auf SAS übertragen.

Nachdem die drei Tabellen des DIMDI nun in SAS importiert sind, soll mit diesen eine relationale Datenbank erstellt werden. Dazu werden mit PROC SQL Tabellen generiert und gleichzeitig die Integritätsbedingungen festgelegt. Der folgende Code zeigt den Aufbau der Datenbank mit PROC SQL:

```
PROC SQL;
     CREATE TABLE ICD10 (Code10 CHAR(7), Titel10 CHAR(255),
         CONSTRAINT Code10X PRIMARY KEY(Code10));
         INSERT INTO ICD10
         SELECT * FROM import_ICD10 WHERE Code10 > "";
     CREATE TABLE ICD9 (Code9 CHAR(7), Titel9 CHAR(255),
         CONSTRAINT Code9X PRIMARY KEY(Code9));
         INSERT INTO ICD9
         SELECT * FROM import_ICD9 WHERE Code9 > "";
     CREATE TABLE Umsteiger (Code10 CHAR(7), Code9 CHAR(7),
         Automatik CHAR1),
         CONSTRAINT UICD10 FOREIGN KEY(Code10) REFERENCES ICD10,
         CONSTRAINT UICD9 FOREIGN KEY(Code9) REFERENCES ICD9);
         INSERT INTO Umsteiger
         SELECT * FROM import_Umsteiger WHERE Code10 > "";
QUIT;
```

Hier werden zunächst mit drei CREATE TABLE Anweisungen neue Tabellen aus den zuvor importierten Tabellen generiert. Dabei wird bei den ersten beiden Tabellen jeweils als Integritätsbedingung (Constraint) die Variable *Code9* bzw. *Code10* als Primärschlüssel festgelegt; die Integritätsbedingungen erhalten dabei die Namen *Code9X* bzw. *Code10X*. Bei der Umsteigertabelle werden dann mit zwei weiteren Integritätsbedingungen die Variablen *Code10* und *Code9* als Fremdschlüssel festgelegt und damit auf die Tabellen *ICD10* und *ICD9* referenziert. Diese Integritätsbedingungen erhalten die Namen *UICD10* und *UICD9*. Die folgende Abbildung zeigt skizzenhaft die daraus folgenden Beziehungen zwischen den Tabellen.

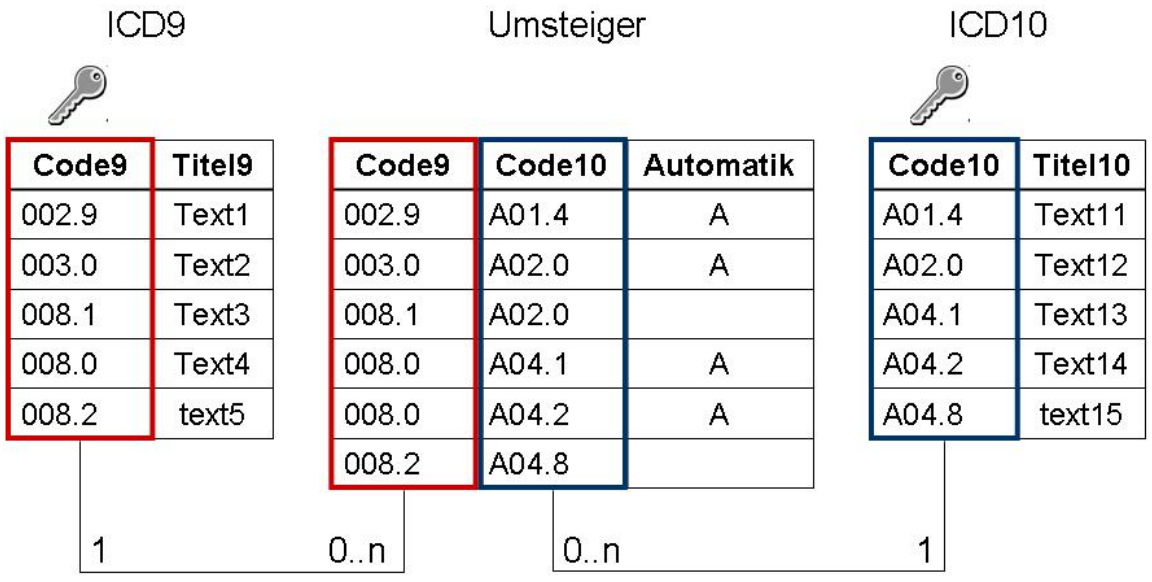

Abbildung 3: Darstellung der Beziehungen zwischen den Tabellen in der relationalen Datenbank

Die beiden Tabellen *ICD9* und *ICD10* besitzen nun eindeutige Primärschlüssel von denen es jeweils 0 bis n Einträge als Fremdschlüssel in der Tabelle Umsteiger geben kann. Damit ist in diesem einfachen Beispiel der Aufbau der relationalen Datenbank abge-

schlossen und diese kann nun für die Umkodierung von ICD9 nach ICD10 verwendet werden.

4 Umkodieren von ICD9-Daten nach ICD10

Mit Hilfe der oben erstellten relationalen Datenbank sollen ICD9-kodierte Todesursachen in ICD10 überführt werden. Auch dies geschieht wieder mit einer PROC SQL-Anweisung:

```
PROC SQL;
     CREATE UNIQUE INDEX _id ON Data_ICD9 (_id);
     CREATE TABLE Ergebnis AS
         SELECT B.*, U.code10, I.titel10
         FROM Data_ICD9 AS B LEFT JOIN
             (Umsteiger AS U LEFT JOIN ICD10 AS I
               ON U.code10 = I.code10) ON B.cause = U.code9
quit;
```

Dabei enthält die Tabelle *Data_ICD9* die festgestellten patientenbezogenen Todesursachen nach ICD9. In dieser Tabelle wird zunächst jedem Eintrag eine eindeutige ID zugewiesen, welche später für die weitere Bearbeitung multipler Zuweisungen benötigt wird (s. Code unten). Danach wird eine Tabelle *Ergebnis* durch eine Verknüpfung der Tabellen *Umsteiger*, *Data_ICD9* und *ICD10* erstellt. Dazu erfolgt zunächst in der inneren Klammer ein LEFT JOIN von der Tabelle *Umsteiger* auf *ICD10*, wodurch jedem Eintrag in der Tabelle *Umsteiger* der Beschreibungstext des jeweiligen ICD10-Codes zugewiesen wird. Die daraus resultierende Tabelle wird dann (ebenfalls wieder über einen LEFT JOIN) mit der Tabelle *Data_ICD9* durch den Abgleich der ICD9-Codes in den Daten mit den ICD9-Codes in der Tabelle *Umsteiger* verknüpft. Durch den LEFT JOIN wird gewährleistet, dass in der Ergebnistabelle auch festgestellte Todesursachen enthalten sind, für die keine Entsprechung in den Tabellen *Umsteiger* und *ICD10* gefunden wurden. Das kann z.B. hilfreich sein, um Datenfehler in der Tabelle *Data_ICD9* aufzudecken. Die Ergebnistabelle hat folgendes Aussehen:

Tabelle 1: Ergebnistabelle der Umkodierung

_ID	Code9	Code10	Titel10	…
1	573.3	K75.2	Unspez. reaktive Hepatitis	…
1	573.3	K75.3	Granulomatöse Hepatitis	…
4	571.5	K76.1	Chronische Stauungsleber	…
8	5999			…
15	573.8	K76.1	Chronische Stauungsleber	…

Die Ergebnistabelle (s.. Tabelle 1) besteht in der linken Hälfte aus den Daten mit den patientenbezogenen Todesursachen und in der rechten Hälfte aus den zugewiesenen

ICD10-Codes mit der entsprechenden Beschreibung. In dieser Tabelle gibt es hier nun zwei ICD9-Codes (_ID 4 und 15) denen derselbe ICD10-Code zugewiesen wird. Allerdings gibt es auch einen patientenbezogenen ICD9-Code, dem mehrere ICD10-Codes zugewiesen werden (_ID 1) und für den daher mehrere Einträge existieren. Durch den LEFT JOIN wurde auch _ID 8 mit in die Tabelle übernommen; dabei handelt es sich anscheinend um einen Kodierungsfehler in der Daten-Tabelle.

Um die multiplen Einträge bei manchen Patienten nun für die Auswertung eindeutig einem ICD10-Code zuzuweisen, sind mehrere Strategien denkbar. Zunächst können über den vorher generierten Index die multiplen Einträge von den eindeutig zugewiesenen Einträgen mit einer PROC SQL-Anweisung getrennt werden:

```
PROC SQL;
     CREATE TABLE multiple_Zuweisungen AS
         SELECT * FROM Ergebnis WHERE _id IN
             (SELECT _id FROM Ergebnis
               GROUP BY _id
               HAVING count(_id > 1))
QUIT;
```

Dabei werden mit `HAVING count (_id >1)` die Anzahl der Einträge pro _ID (Gruppierung erzwungen durch die GROUP-Anweisung davor) gezählt und nur Einträge mit mehreren Zuweisungen in die Tabelle *multiple_Zuweisungen* übernommen. Für die endgültigen Zuweisungen in dieser Tabelle könnte z.B. für ICD9-Codes jeweils nur die oberste ICD10-Gruppe aus den möglichen Zuweisungen ausgewählt werden. Oder man definiert für jeden ICD9-Code einen festen ICD10-Code und ordnet diesen alle gleichen ICD9-Codes in dieser Tabelle zu.

5 Löschen der relationalen Datenbank

Die erstellte relationale Datenbank ist in der SAS-Arbeitsumgebung in Form von drei "normal" aussehenden Data Sets sichtbar. Diese drei Data Sets sind allerdings durch die festgelegten Integritätsbedingungen miteinander verbunden und können daher nicht wie übliche Data Sets einfach gelöscht oder verändert werden. Die Integritätsbedingungen der bestehenden Tabellen können mit einer PROC SQL-Anweisung angezeigt werden:

```
PROC SQL;
     DESCRIBE TABLE CONSTRAINTS Umsteiger;
quit;
```

Hiermit werden die Namen der Integritätsbedingungen und ihre Bedingung selbst - hier für die Tabelle *Umsteiger* - ausgegeben. Um nun die relationale Datenbank in SAS löschen zu können, müssen zunächst die Integritätsbedingungen zwischen und in den Tabellen schrittweise entfernt werden. Dabei ist vor allem darauf zu achten, dass die

Fremdschlüssel vor den Primärschlüsseln entfernt werden. Erst danach können die Tabellen gelöscht werden.

```
PROC SQL;
     ALTER TABLE Umsteiger
     DROP CONSTRAINT UICD10, UICD9;
     ALTER TABLE ICD9
     DROP CONSTRAINT Code9X;
     ALTER TABLE ICD10
     DROP CONSTRAINT Code10X;
     DROP TABLE ICD9, ICD10, Umsteiger;
quit;
```

Die ALTER-Anweisung erlaubt Änderungen in den Tabellen durchzuführen, in diesem Fall wird die Integritätsbedingung jeweils mit DROP aus der Tabelle entfernt und diese dadurch verändert. Die Integritätsbedingungen müssen dabei entsprechend der beim Aufbau der relationalen Datenbank festgelegten Namen angesprochen werden. Die letzte DROP-Anweisung entfernt letztendlich die Tabellen aus der SAS Arbeitsumgebung.

Literatur

[1] C. Dickstein, et.al. (2004): DATA Step vs. PROC SQL: What's a neophyte to do? SUGI 29 Tutorials, Paper 269-29, Princeton, New Jersey

[2] M. Weires (2008): Einführung in PROC SQL. www.urz.uni-heidelberg.de/imperia/md/content/urz/programme/statistik/sas-treff/2008-07-10.pdf [Zugriff 15.03.2011]

[3] www.urz.uni-heidelberg.de/statistik/sas-ah/01.01.02/maske.html [Zugriff 15.03.2011]

[4] ICD9-ICD10-Überleitungstabelle von 2001: www.dimdi.de/dynamic/de/klassi/downloadcenter/icd-10-who/vorgaenger/version10/ueberleitung/ [Zugriff 16.03.2011; neuere Versionen sind kostenpflichtig!]