JOIN und MERGE: Zwei Seiten einer Tabellenverknüpfung

Marianne Weires
Deutsches Krebsforschungszentrum (DKFZ)
Im Neuenheimer Feld 580
69120 Heidelberg
m.weires@dkfz.de

Zusammenfassung

SQL steht für Structured Query Language und ist eine weit verbreitete Datenbanksprache zur Definition, Abfrage und Bearbeitung von Daten in relationalen Datenbanken. PROC SQL ist die SAS Implementierung von SQL und stellt oft eine Alternative zum SAS Datenschritt dar. So werden z.B. in PROC SQL Tabellen mit Hilfe des JOIN Befehls, ähnlich zum MERGE im Datenschritt, miteinander verknüpft und Beziehungen zwischen Tabellen aufgezeigt.
Obwohl in der Literatur JOIN oft als das Gegenstück zum MERGE angesehen wird, können beide in bestimmten Situationen durchaus unterschiedliche Ergebnisse liefern. In diesem Beitrag werde ich gezielt solche Situationen aufzeigen und das unterschiedliche Verhalten erläutern. Zusätzlich möchte ich auch auf die verschiedenen JOIN-Typen, wie z.B. NATURAL, CROSS und OUTER JOIN in PROC SQL eingehen und deren Einsatz mit Beispielen veranschaulichen.

Schlüsselwörter: PROC SQL, JOIN, OUTER JOIN, INNER JOIN, NATURAL JOIN, MERGE, Self-Join

1 Tabellenverknüpfung mit JOIN oder MERGE

Mit den JOIN Befehlen in PROC SQL können zwei oder mehrere Tabellen oder Views ähnlich zum MERGE im Datenschritt (horizontal) miteinander verknüpft werden und so Beziehungen zwischen Tabellen aufgezeigt werden. Obwohl JOIN oft als das Gegenstück zu MERGE angesehen wird, können beide in bestimmten Situationen unterschiedliche Ergebnisse liefern. Im Folgenden wird zunächst die allgemeine Syntax eines JOINs erläutert. Anschließend werden anhand von ausgewählten Szenarien JOIN und MERGE miteinander verglichen und sowohl die Gemeinsamkeiten als auch die Unterschiede rausgestellt. Wenn möglich werde ich aufzeigen wie der JOIN Befehl an die Ausgabe des entsprechenden MERGE Befehls angepasst werden kann, so dass beide die gleiche Ausgabe liefern. Als letzten Punkt werde ich zusätzlich noch die besonderen JOIN-Typen, wie den Non-Equijoin, NATURAL JOIN und den Self-Join erläutern.

2 Syntax eines JOINs in PROC SQL

Im Folgenden ist die allgemeine Syntax eines JOINs für zwei Tabellen/Views in PROC SQL dargestellt.

```
PROC SQL <Optionen>;
  CREATE TABLE ErgebnisTabelle AS
    SELECT ...
      FROM Tab1|View1 <<AS> t1> <JOIN-Typ> Tab2|View2 <<AS> t2>
      <ON (JOIN-Bedingung)> ;
QUIT;
```

PROC SQL ist eine interaktive Prozedur und bleibt bis zu einem QUIT aktiv. Es können mehrere SQL Anweisungen, getrennt durch einen Strichpunkt, innerhalb eines SQL Blocks spezifiziert werden. Die Tabellenverknüpfung wird innerhalb der SELECT Anweisung kodiert. Für jede Tabelle kann ein Alias mit oder ohne AS und Kürzel angegeben werden. Der JOIN-Typ gibt die Art der Verknüpfung an z.B. CROSS, INNER oder OUTER JOIN. Die JOIN-Bedingung definiert die JOIN-Kriterien, d.h. wie und über welche Variablen (hier: Schlüsselvariablen) die Tabellen verknüpft werden sollen. Je nach Kriterium kann jeder JOIN-Typ noch zusätzlich als Equi- und non-Equijoin kategorisiert werden. Ein Equijoin basiert auf einem "=" in der JOIN-Bedingung. Non-Equijoins basieren, wie die Bezeichnung schon andeutet, auf einem "nicht =" oder auf der Verbindung von mehreren JOIN-Bedingungen mit einem OR. Mit Hilfe des CREATE TABLE ... AS Befehls kann aus der Ausgabe (Ergebnis) eines JOINs eine neue Tabelle erstellt werden.

3 Tabellenverknüpfung ohne Schlüsselvariablen

Sowohl ein JOIN als auch ein MERGE können ohne eine Bedingung spezifiziert und ausgeführt werden, d.h. ein JOIN wird ohne eine ON-Anweisung (JOIN-Bedingung) und ein MERGE ohne eine BY-Anweisung spezifiziert. Dieses einfache Beispiel wird bereits das unterschiedliche Verhalten von MERGE und JOIN veranschaulichen. Zur Demonstration verwende ich zwei ganz einfache Datensätze eins und zwei mit jeweils einer Variablen name bzw vorname:

Tabelle 1: Beispieltabelle eins

name
Louise
John
Alice

Tabelle 2: Beispieltabelle zwei

vorname
Robert
Louise
John

Zunächst der MERGE Befehl ohne BY-Anweisung und die dazugehörige Ausgabe.

```
DATA combined_merge;
MERGE eins zwei;
RUN;
```

Tabelle 3: Ausgabe MERGE ohne BY-Anweisung

name	vorname
Louise	Robert
John	Louise
Alice	John

Dann der JOIN Befehl ohne ON-Anweisung und die dazugehörige Ausgabe.

```
PROC SQL;
  CREATE TABLE combined_join AS
    SELECT *
      FROM eins CROSS JOIN zwei;

  /*analog auch möglich*/
  CREATE TABLE combined_join AS
    SELECT *
      FROM eins,zwei;
QUIT;
```

Tabelle 4: Ausgabe CROSS JOIN ohne ON-Anweisung

name	vorname
Louise	Robert
Louise	Louise
Louise	John
John	Robert
John	Louise
John	John
Alice	Robert
Alice	Louise
Alice	John

Es ist sofort zu erkennen, dass MERGE und JOIN völlig unterschiedliche Ergebnisse liefern. Damit lässt sich schon das unterschiedliche Vorgehen von MERGE und JOIN

erkennen. Während beim MERGE eine Beobachtung der Tabelle eins mit jeweils einer einzigen Beobachtung von zwei gepaart wird (one-to-one Verknüpfung), werden beim CROSS JOIN alle Beobachtungen der Tabelle eins mit allen Beobachtungen der Tabelle zwei verknüpft. Diese vollständige Kombination wird auch Kartesisches Produkt genannt.

Hinweis: Aufgrund der immensen Kombinationsmöglichkeiten ist dieser JOIN besonders bei größeren Tabellen sehr rechenintensiv und sollte nach Möglichkeit vermieden werden. Es wird auch ein Hinweis im Log-Fenster angezeigt: `NOTE: The execution of this query involves performing one or more Cartesian product joins that can not be optimized.`

4 Tabellenverknüpfung mit eindeutigen Werten der Schlüsselvariablen

Im Beispiel bisher wurden die Tabellen ohne die Angabe einer Schlüsselvariablen verknüpft. Nun soll das Verhalten von MERGE und JOIN verglichen werden, wenn zwei Tabellen basierend auf einer Schlüsselvariablen verknüpft werden. Die Werte der Schlüsselvariablen sind jeweils eindeutig in beiden Tabellen. Zunächst wird ein MERGE mit BY-Anweisung und ohne IN-Filter demonstriert und der dazugehörige FULL JOIN (ein OUTER JOIN).

Tabelle 5: Beispieltabelle eins mit eindeutigen Werten der Schlüsselvariablen name

name	gewicht
Louise	51
John	89
Alice	43

Tabelle 6: Beispieltabelle zwei mit eindeutigen Werten der Schlüsselvariablen vorname

vorname	groesse
Robert	172
Louise	152
John	173

Der MERGE Befehl (mit vorheriger Sortierung) und die dazugehörige Ausgabe.

```
PROC SORT DATA=eins OUT=eins_s;BY name;
PROC SORT DATA=zwei OUT=zwei_s;BY vorname;

DATA combined_merge;
MERGE eins_s zwei_s(rename=(vorname=name));
BY name;
RUN;
```

Tabelle 7: Ausgabe MERGE

name	gewicht	groesse
Alice	43	.
John	89	173
Louise	51	152
Robert	.	172

```
PROC SQL;
  CREATE TABLE combined_join AS
    SELECT *
      FROM eins FULL JOIN zwei
      ON name=vorname;
QUIT;
```

Tabelle 8: Ausgabe FULL JOIN

name	gewicht	vorname	groesse
Alice	43		.
John	89	John	173
Louise	51	Louise	152
	.	Robert	172

Im Gegensatz zum JOIN müssen beim MERGE zunächst die Tabellen sortiert werden (oder es liegt ein passender Index vor) und eventuell müssen die Schlüsselvariablen mit RENAME unbenannt werden. Im Ergebnis von MERGE wird die gleichnamige Schlüsselvariable name überlagert und taucht auch nur einmal auf. In der Ausgabe von JOIN tauchen beide Schlüsselvariablen, name und vorname, auf. Mit Hilfe der COALESCE Funktion kann die Ausgabe an die von MERGE angepasst werden. Diese Funktion nimmt den ersten nicht-leeren Wert in einer Liste.

```
PROC SQL;
  CREATE TABLE combined_join2(DROP=name vorname) AS
    SELECT COALESCE(name,vorname) AS nameNeu,*
      FROM eins FULL JOIN zwei
      ON name=vorname;
QUIT;
```

Tabelle 9: Ausgabe FULL JOIN mit COALESCE Funktion

nameNeu	gewicht	groesse
Alice	43	.
John	89	173
Louise	51	152
Robert	.	172

5 Tabellenverknüpfung mit eindeutigen und wiederholenden Werten der Schlüsselvariablen (one-to-many Verknüpfung)

In den Beispielen vorher waren die Werte der Schlüsselvariablen in beiden Tabellen eindeutig. Nun sind in einer der Tabellen die Werte eindeutig und in der anderen Tabelle wiederholen sich die Werte (one-to-many Verknüpfung). Es wird die Ausgaben von MERGE mit BY-Anweisung und IN-Filter zum dazugehörigen RIGHT JOIN (ein OUTER JOIN) verglichen.

Tabelle 10: Beispieltabelle eins mit eindeutigen Werten der Schlüsselvariablen name

name	gewicht
Louise	51
John	89
Alice	56

Tabelle 11: Beispieltabelle zwei mit wiederholenden Werten der Schlüsselvariablen name

name	groesse
Robert	172
Louise	152
John	173
John	190

Der MERGE Befehl mit IN-Filter (rechte Tabelle) und die dazugehörige Ausgabe.

```
DATA combined_merge;
MERGE eins_s zwei_s(IN=in_zwei);
BY name;
IF in_zwei;
RUN;
```

Tabelle 12: Ausgabe von MERGE

name	gewicht	groesse
John	89	173
John	89	190
Louise	51	152
Robert	.	172

Die Verknüpfung mit RIGHT JOIN und die dazugehörige direkte Ausgabe mit SELECT.

```
PROC SQL;
  SELECT *
    FROM eins AS e RIGHT JOIN zwei z
    ON e.name=z.name;
QUIT;
```

Tabelle 13: Ausgabe von RIGHT JOIN (SELECT)

name	gewicht	name	groesse
John	89	John	173
John	89	John	190
Louise	51	Louise	152
	.	Robert	172

Wie bereits in den vorherigen Beispiel beobachtet man, dass die Schlüsselvariablen nicht wie beim MERGE überlagert werden, d.h. name taucht zweimal in der Ausgabe auf. Zu beachten ist jedoch, falls eine neue Tabelle mit CREATE TABLE aus der Abfrage erstellt wird. In diesem Fall wird standardmäßig die ganz linke/erste Spalte von name (eins.name) beibehalten. Es wird ein Warnhinweis im Log Fenster angezeigt: `WARNING: Variable name already exists on file WORK.COMBINED_JOIN.` Die Ausgabe ist weniger sinnvoll, denn nun fehlt der Eintrag für "Robert" in name und es wird ein leeres Feld angezeigt:

```
PROC SQL;
  CREATE TABLE combined_join AS
    SELECT *
      FROM eins AS e RIGHT JOIN zwei z
      ON e.name=z.name;
QUIT;
```

Tabelle 14: Ausgabe von RIGHT JOIN (CREATE TABLE)

name	gewicht	groesse
John	89	173
John	89	190
Louise	51	152
	.	172

Besser ist es die letzte/rechte Variable zwei.name gezielt zu selektieren und umzubenennen oder wieder mit der COALESCE Funktion zu arbeiten:

```
PROC SQL;
  CREATE TABLE combined_join3 AS
    SELECT z.name AS nameNeu,e.gewicht,z.groesse
      FROM eins AS e RIGHT JOIN zwei z
      ON e.name=z.name;
QUIT;
PROC SQL;
  CREATE TABLE combined_join2 AS
    SELECT COALESCE(e.name,z.name) AS nameNeu,e.gewicht,z.groesse
      FROM eins AS e RIGHT JOIN zwei z
      ON e.name=z.name;
QUIT;
```

Tabelle 15: Ausgabe von RIGHT JOIN mit COALESCE

nameNeu	gewicht	groesse
John	89	173
John	89	190
Louise	51	152
Robert	.	172

6 Tabellenverknüpfung mit wiederholenden Werten der Schlüsselvariablen (many-to-many Verknüpfung)

In den Beispielen bisher waren in einer der Tabellen die Werte eindeutig und in der anderen Tabelle nicht (wiederholende Werte). Die Anzahl der zurückgelieferten Beobachtungen mittels MERGE und JOIN war stets gleich. Wiederholen sich die Werte der Schlüsselvariablen jedoch in den beteiligten Tabellen, verhalten sich MERGE und JOIN grundlegend anders. Die Vorgehensweise von MERGE wird kurz erläutert.

Tabelle 16: Beispieltabelle person

name	alter	gewicht
Louise	70	51
Alice	66	56
Emma	43	60
Anna	34	70
Robert	74	90
John	64	92
Richard	35	95
Thomas	36	87
Monica	71	60
Lars	46	80

Tabelle 17: Beispieltabelle diagnose

icd7	krankheit
170	Brustkrebs
157	Magenkrebs
180	Nierenkrebs

Tabelle 18: Beispieltabelle person_diagnose zur Verknüpfung der Tabellen person und diagnose mit wiederholenden Schlüsselwerten

name	icd7
Louise	170
Louise	157
Alice	157
Alice	180
Emma	170
Anna	180

Tabelle 19: Beispieltabelle stammbaum mit wiederholenden Schlüsselwerten

vater	mutter	kind
Robert	Louise	Emma
Robert	Louise	Richard
Robert	Louise	Thomas
John	Alice	Anna
John	Monica	Lars

Wenn man nun zu jeder Mutter herausfinden möchte, welche Diagnose die Tochter hat, kann dies mit JOIN so realisiert werden: Zunächst verknüpfen wir die Tabellen stammbaum und person_diagnose und erhalten Mütter und ihre Diagnose.

```
PROC SQL;
  CREATE TABLE mutter_diag AS
    SELECT mutter,icd7,kind
      FROM person_diagnose INNER JOIN stammbaum
      ON mutter=name;
QUIT;
```

Tabelle 20: Inhalt der Tabelle mutter_diag

mutter	icd7	kind
Louise	170	Emma
Louise	170	Thomas
Louise	170	Richard
Louise	157	Emma
Louise	157	Thomas
Louise	157	Richard
Alice	157	Anna
Alice	180	Anna

Dann verwenden wir die Tabelle mutter_diag und verknüpfen diese mit der Tabelle person_diagnose, um die Diagnose der Kinder zu erhalten.

```
PROC SQL;
    SELECT mutter,m.icd7 AS icd7m,m.kind,p.icd7 AS icd7k
      FROM mutter_diag m INNER JOIN person_diagnose p
      ON kind=name;
QUIT;
```

Tabelle 21: Ausgabe von JOIN

mutter	icd7m	kind	icd7k
Louise	170	Emma	170
Louise	157	Emma	170
Alice	157	Anna	180
Alice	180	Anna	180

Versucht man nun analog mit MERGE die Tabelle mutter_diag zu erstellen, erhält man folgenden Hinweis im Log-Fenster: `NOTE: MERGE statement has more than one data set with repeats of BY values.`

```
DATA mutter_diag_merge;
   MERGE person_diagnose_s(IN=in_eins)
      stammbaum_s(RENAME=(mutter=name) IN=in_zwei DROP=vater);
   BY name;
   IF in_eins AND in_zwei;
RUN;
```

Tabelle 22: Ausgabe von MERGE

name	icd7	kind
Alice	157	Anna
Alice	180	Anna
Louise	170	Emma
Louise	157	Richard
Louise	157	Thomas

Es ist sofort zu erkennen, dass MERGE und JOIN grundsätzlich andere Ergebnisse liefern. Beim JOIN werden alle Beobachtungen aus Tabelle person_diagnose zu allen passenden Beobachtungen in Tabelle stammbaum verknüpft (anhand der übereinstimmenden Schlüsselwerte). Ein ähnliches Verhalten konnte schon bei der Tabellenverknüpfung ohne Schlüsselvariablen beobachtet werden.
Beim MERGE wird jeweils eine Beobachtung aus Tabelle person_diagnose zeilenweise mit einer Beobachtung aus Tabelle stammbaum gepaart (one-to-one Verknüpfung anhand der übereinstimmenden Schlüsselwerte). Die Vorgehensweise von MERGE kann mit dem Program Data Vector (PDV) nachvollzogen werden (fett gedruckt ist immer der veränderte Variablenwert):

1. Aus Tabelle person_diagnose_s wird die erste Beobachtung von „Alice“ in den leeren PDV eingetragen

name	icd7	kind
Alice	157	

Aus Tabelle stammbaum wird nun die Beobachtung von „Alice“ in den PDV eingetragen und dann der Inhalt ausgegeben.

name	icd7	kind
Alice	157	**Anna**

→ Ausgabe

2. Nun wird aus Tabelle person_diagnose_s die zweite Beobachtung von „Alice“ in den PDV eingetragen und überschreibt den Inhalt der Variable icd7. Der Inhalt der Variable kind bleibt erhalten d.h. wird „vererbt“.

name	icd7	kind
Alice	**180**	Anna

→ Ausgabe

3. Da es keine weiteren Beobachtungen von „Alice“ gibt, wird der PDV neu initialisiert und mit Werten der neuen BY-Gruppe „Louise“ gefüllt.

name	icd7	kind
Louise	**170**	

4. Aus der Tabelle stammbaum_s wird die erste Beobachtung von „Louise“ in den PDV eingetragen. Variable kind wird gefüllt.

name	icd7	kind
Louise	170	**Emma**

→ Ausgabe

5. Da es jeweils noch weitere Beobachtungen von „Louise“ in Tabelle person_diagnose_s und stammbaum_s gibt, werden die Werte aus person_diagnose_s eingetragen.

name	icd7	kind
Louise	**157**	Emma

5. Dann werden die Werte aus der Tabelle stammbaum_s eingetragen.

name	icd7	kind
Louise	157	**Richard**

→ Ausgabe

6. Da es keine weitere Beobachtung „Louise“ in Tabelle person_diagnose_s, werden die Werte der Variablen name und icd7 beibehalten und die Werte aus der Tabelle stammbaum_s eingetragen. Die Variable kind wird neu gefüllt mit dem Wert "Thomas".

name	icd7	kind
Louise	157	**Thomas**

→ Ausgabe

Das Ergebnis ist in Tabelle 22 zu sehen.

Hinweis: Den aufmerksamen Leser mag aufgefallen sein, dass das Ergebnis in den Tabellen 20 und 21 (INNER JOIN) in diesem Beispiel nicht mehr sortiert ist. Der Grund

hierfür ist, dass JOINs algorithmisch anders umgesetzt werden wie MERGE. Die Umsetzungsalgorithmen sind: hash join, sort merge join, index join und step loop join.

7 Weitere JOIN-Typen

Im Folgenden möchte ich noch auf weitere JOIN-Typen eingehen, die kein direktes Gegenstück mit MERGE haben:

- Non-Equijoin
- NATURAL JOIN
- Self-Join

7.1 Non-Equijoin

Während bei einem MERGE die Verknüpfung stets über die Gleichheit der Werte der Schlüsselvariablen erfolgt, kann ein JOIN aber auch über die Ungleichheit von Schlüsselvariablen verknüpfen, man spricht dann von einem Non-Equijoin (im Gegensatz zum Equijoin).

Tabelle 23: Beispieltabelle eins

name	alter	gewicht
Louise	41	51
John	52	89
Alice	43	56

Tabelle 24: Beispieltabelle zwei

vorname	alter	groesse
Robert	28	172
Louise	77	152
John	52	173

Ein Non-Equi-INNER JOIN sieht dann folgendermaßen aus.

```
PROC SQL;
  SELECT e.*,z.*
    FROM eins AS e INNER JOIN zwei z
    ON e.name<z.name;
QUIT;
```

Tabelle 25: Ausgabe Non-Equi-INNER JOIN

name	alter	gewicht	vorname	alter	groesse
Louise	41	51	Robert	28	172
John	52	89	Robert	28	172
John	52	89	Louise	77	152
Alice	43	56	Robert	28	172
Alice	43	56	Louise	77	152
Alice	43	56	John	52	173

Es werden nur die Beobachtungen zurückgegeben, für die die JOIN-Bedingung „wahr“ ist, d.h. `e.name<z.name`.

Hinweis: Im Vergleich zu Equijoins sind Non-Equijoins weniger effizient, da sie durch ein Kartesisches Produkt umgesetzt werden (siehe auch den entsprechenden Hinweis im Log-Fenster). Equijoins können hingegen optimiert werden.

7.2 NATURAL JOIN

Ein NATURAL JOIN verknüpft die Tabellen über die Gleichheit aller jeweils gleichnamigen und gleichtypigen Spalten (d.h. die JOIN-Bedingung ist implizit). Die gleich benannten Spalten aus den Tabellen werden im Ergebnis nur einmal angezeigt. Gibt es keine gleichnamigen Spalten, wird der NATURAL JOIN zum CROSS JOIN. Mit der Anweisungsoption FEEDBACK, werden die JOIN-Bedingung und der JOIN-Typ im Log-Fenster angezeigt.

```
PROC SQL FEEDBACK;
  SELECT *
    FROM eins NATURAL JOIN zwei;
QUIT;

NOTE: Statement transforms to:
select COALESCE(ZWEI.alter, EINS.alter) as alter, ZWEI.vorname,
ZWEI.groesse,EINS.name,EINS.gewicht from WORK.EINS inner join
WORK.ZWEI on ZWEI.alter = EINS.alter;
```

7.3 Self-Join

Bei dieser Art von JOIN wird eine Tabelle mit sich selbst verknüpft. Dieser JOIN wird hauptsächlich dazu benutzt, um Beziehungen innerhalb einer Tabelle zu veranschaulichen. PROC SQL macht eine Kopie der Tabelle und verknüpft die Tabelle mit dieser Kopie. Dieser JOIN wird anhand der Beispieltabelle mitarbeiter veranschaulicht.

Tabelle 26: Beispieltabelle mitarbeiter

id	name	boss_id
1	Louise	5
2	John	1
3	Alice	1
4	Albert	2
5	Robert	.

Der Self-Join und die dazugehörige Ausgabe. Das Ergebnis sind alle Mitarbeiter die einen Boss haben und die boss_id wurde durch den dazugehörigen Namen ersetzt. Da mit derselben Tabelle gearbeitet wird, muss an dieser Stelle immer ein Tabellenalias verwendet werden.

```
PROC SQL;
  SELECT mitar.name AS name_mitarbeiter,
```

```
        vorge.name "name_vorgesetzter"
    FROM mitarbeiter AS mitar INNER JOIN mitarbeiter AS vorge
    ON mitar.boss_id = vorge.id;
QUIT;
```

Tabelle 27: Ausgabe Self-Join

name_mitarbeiter	name_vorgesetzter
Louise	Robert
John	Louise
Alice	Louise
Albert	John

Der Self-Join kann aber auch als LEFT JOIN formuliert werden. Nun werden zusätzlich zu der obigen Ausgabe auch alle Mitarbeiter zurückgeliefert, die keinen Boss haben. Es kann wieder mit der Funktion COALESCE gearbeitet werden.

```
PROC SQL;
  SELECT mitar.name AS name_mitarbeiter,
         COALESCE(vorge.name,"hat keinen Boss")
         "name_vorgesetzter"
     FROM mitarbeiter AS mitar LEFT JOIN mitarbeiter AS vorge
     ON mitar.boss_id = vorge.id;
QUIT;
```

Tabelle 28: Ausgabe Self-Join und COALESCE

name_mitarbeiter	name_vorgesetzter
Robert	hat keinen Boss
Alice	Louise
John	Louise
Albert	John
Louise	Robert

8 Zusammenfassung

Sowohl MERGE als auch JOIN ermöglichen es, Tabellen horizontal miteinander zu verknüpfen. Obwohl die standardmäßige Ausgabe von MERGE und JOIN unterschiedlich ist, kann die Ausgabe von JOIN in manchen Situation an die von MERGE angepasst werden. Trotzdem sollte beachtet werden, dass sich MERGE und JOIN in ihrer Vorgehensweise unterscheiden. Während beim MERGE Beobachtungen one-to-one miteinander gepaart werden, wird beim JOIN jede Beobachtung mit jeder (passenden) Beobachtung kombiniert. Dies kann zu sehr unterschiedlichen Ergebnissen führen, wie etwa bei many-to-many Verknüpfungen. Darüber hinaus bietet PROC SQL auch weitere JOIN-Typen an, wie etwa den Non-Equijoin, NATRUAL JOIN und Self-Join.