SQL Processing mit der SAS Software: Eine Einführung in die Prozedur SQL

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

Zusammenfassung

In diesem Beitrag wird eine Einführung in PROC SQL gegeben und zugleich wird die Mächtigkeit von SQL aufgezeigt. Neben grundlegenden Befehlen zum Arbeiten mit Tabellen und Views wird die Struktur von SQL Anfragen erläutert. Einen besonderen Schwerpunkt bildet die Verknüpfung von mehreren Tabellen, die mittels PROC SQL oft einfach und übersichtlich zu lösen ist. Geschachtelte Anfragen werden ebenso behandelt wie die Verknüpfung von mehreren Anfragen mittels der SET-Operatoren. Zusätzlich wird die Erstellung von Makrovariablen in PROC SQL gezeigt und es wird auf Indizes eingegangen, die eine effiziente Suche in großen Datenmengen ermöglichen. Zum Schluss wird die Anbindung relationaler Datenbanksysteme mit SAS/ACCESS vorgestellt.

Schlüsselwörter: PROC SQL, SELECT, Views, Join, INNER JOIN, OUTER JOIN, Subqueries, SET-Operatoren, OUTER UNION, UNION, INTERSECT, EXCEPT, Makrovariable, Index, SAS/ACCESS, Pass-Through Facility, SQL-Prozedur

1 Was ist SQL und PROC SQL

SQL steht für **S**tructured **Q**uery **L**anguage und ist eine standardisierte und weit verbreitete Datenbanksprache zur Definition, Anfrage und Bearbeitung von Daten in relationalen Datenbanken. SQL ist seit 1986 und 1987 ein ANSI- bzw. ISO-Standard, der bereits mehrfach spezifiziert wurde bzw. noch wird. Neben einem bestimmten Standard unterstützen Datenbanksysteme meist Teile höherer Standards sowie eigene SQL-Erweiterungen. PROC SQL ist die SAS Implementierung von SQL und stellt oft eine Alternative zum SAS Datenschritt dar. Seit SAS Version 6 ist die Prozedur Teil der SAS-Base Software. Der Einsatz von PROC SQL ist sehr vielfältig, so können folgende Operationen sehr einfach und übersichtlich umgesetzt werden:

- Erstellung von Summenstatistiken
- Erzeugung von Tabellen, Views und Indizes
- Verknüpfung von Tabellen oder Views
- Erzeugung von Makrovariablen
- Abfrage externer Datenbanken

Tabelle 1: SQL vs SAS Datenschritt

SQL	SAS Datenschritt
Tabelle	SAS Datendatei
Zeile	Beobachtung
Spalte	Variable
Join	Merge

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. Im Folgenden werden die Beispieltabellen vorgestellt.
sql.mitarbeiter: Daten von 15 Mitarbeitern, die alle eine eindeutige Id haben.
sql.projekt: Daten von 6 Projekten, die alle eine eindeutige Id haben.
sql.zuordnung: Zuordnung von Mitarbeitern zu Projekten und ihre Rolle.

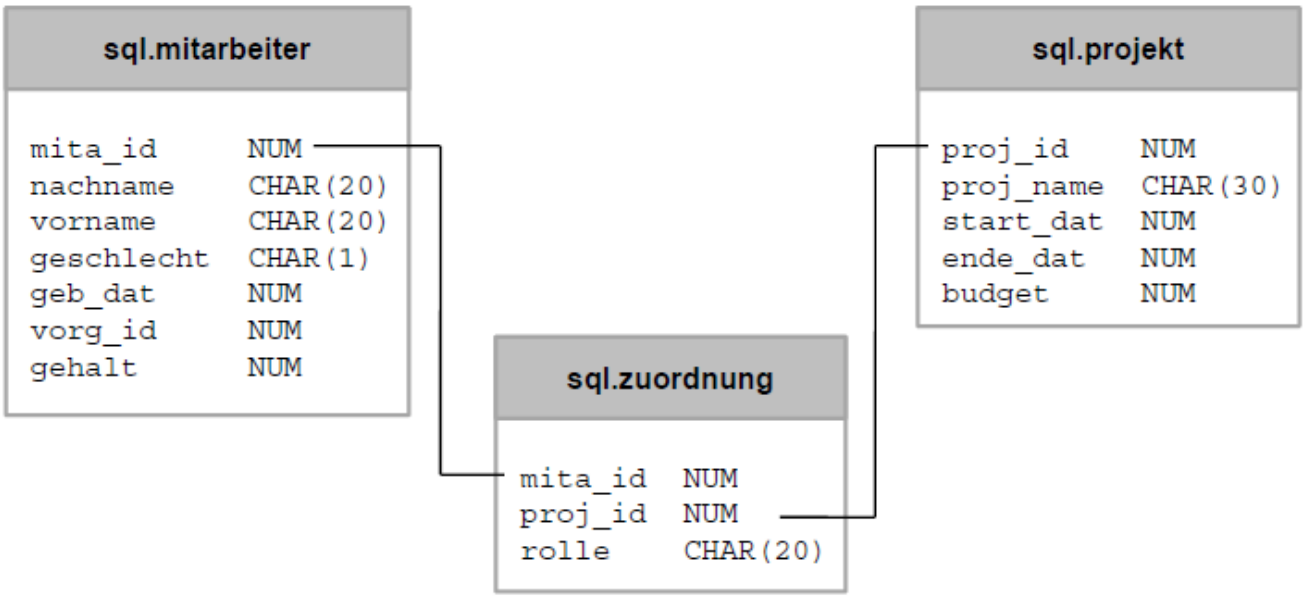

Abbildung 1: Beispieltabellen

2 Arbeiten mit Tabellen

2.1 Erstellen, Verändern und Löschen von Tabellen

Es gibt drei Wege mit Hilfe von CREATE TABLE eine neue Tabelle zu erstellen.
1) CREATE TABLE: Es wird eine leere Tabelle aus Spaltendefinitionen erstellt.

```
PROC SQL;
  CREATE TABLE sql.projekt(
    proj_id NUM LABEL='projekt',
    proj_name CHAR(30) LABEL='projekt bezeichnung',
    start_dat NUM LABEL='anfang des projektes' FORMAT=eurdfwdx.,
    ...
  );
QUIT;
```

LABEL und FORMAT Angaben können mit angegeben werden, sind aber optional.

2) CREATE TABLE...LIKE: Es wird eine leere Tabelle basierend auf der Struktur einer vorhandenen Tabelle erstellt. Dabei entfällt die genaue Definition der einzelnen Spalten.

```
PROC SQL;
  CREATE TABLE sql.projekt_kopie LIKE sql.projekt;
QUIT;
```

3) CREATE TABLE...AS: Es wird eine Tabelle basierend auf der Struktur und Inhalt einer vorhandenen Tabelle bzw. Anfrage erstellt.

```
PROC SQL;
  CREATE TABLE sql.projekt_kopie AS
    SELECT *
      FROM sql.projekt;
QUIT;
```

Mit DESCRIBE TABLE wird der dazugehörige CREATE TABLE Befehl im LOG Fenster angezeigt. Mit dem ALTER TABLE...ADD/DROP/MODIFY Befehl können neue Spalten zu einer Tabelle hinzufügt, Spalten gelöscht oder verändert werden. Mit DROP TABLE wird eine Tabelle gelöscht.

```
PROC SQL;
  ALTER TABLE sql.projekt_kopie
    ADD auftraggeber CHAR(20),dringlichkeit NUM
    DROP budget
    MODIFY proj_name CHAR(50);
  DROP TABLE sql.projekt_kopie;
QUIT;
```

2.2 Einfügen, Ändern und Löschen von Beobachtungen

Mit dem INSERT INTO Befehl gibt es drei Möglichkeiten Beobachtungen in eine Tabelle einzugefügen. Die neuen Einträge werden ans Ende der Tabelle angehangen.
1) Einfügen mit der SET Anweisung
2) Einfügen mit der VALUES Anweisung
3) Einfügen aus einer SQL Anfrage

```
PROC SQL;
  INSERT INTO sql.projekt_kopie /*1)*/
    SET proj_id = 108,
        proj_name = 'NeuesProjekt1',
        start_dat = '12JAN2006'd; /*ende_dat und budget leer*/
```

```
  INSERT INTO sql.projekt_kopie /*2)*/
    VALUES(109,'NeuesProjekt3','15MAR2007'd,'26MAR2007'd,45000)
    VALUES(110,'NeuesProjekt4','18JUN2008'd,.,30000);
  INSERT INTO sql.projekt_kopie /*3)*/
    SELECT *
      FROM sql.projekt;
QUIT;
```

Mit UPDATE und DELETE können alle oder ausgewählte Beobachtungen in einer Tabelle geändert bzw. gelöscht werden.

```
PROC SQL;
  UPDATE sql.projekt_kopie
    SET budget=budget*10; /*alle Einträge betroffen*/
  DELETE
    FROM sql.projekt_kopie WHERE proj_id=101;
QUIT;
```

3 SQL Anfragen

Das Kernstück von SQL ist die SELECT Anweisung, mit der Daten aus einer oder mehreren Tabellen selektiert, neue Informationen berechnet oder auch mehrere Tabellen miteinander verknüpft werden können.

3.1 Struktur der SELECT Anweisung

Die Grundstruktur und Reihenfolge der einzelnen Elemente einer SELECT Anweisung sind im Folgenden dargestellt. Die Elemente in spitzen Klammern sind optional.

```
1. SELECT      ausgewählte Spalte(n)
2. FROM        ausgewählte Tabelle(n), View(s)
3. <ON>        Kriterien für join/merge
4. <WHERE>     Filterkriterien für Zeile(n)
5. <GROUP BY>  Gruppierung über diese Spalte(n)
6. <HAVING>    Filterkriterien für Gruppierung(en) oder Kennwerte
7. <ORDER BY>  Sortierung nach dieser Spalte(n)
```

Um die Funktionsweise der einzelnen Elemente besser zu verstehen, sollte an dieser Stelle die Verarbeitungsreihenfolge erwähnt werden: SELECT, WHERE und GROUP BY wirken auf Spalten, wenn diese in die Anfrage „reinkommen“, HAVING und ORDER BY, wenn diese die Anfrage „verlassen“. Mit VALIDATE kann die Syntax einer SQL Anfrage überprüft werden, ohne dass diese ausgeführt wird. Wird SELECT ohne weitere Ergänzungen verwendet, so werden die Daten der Anfrage im OUTPUT Fenster angezeigt.

3.2 Formulieren von SQL Anfragen

3.2.1 Auswählen von Spalten

Mit „*“ werden sämtliche Spalten ausgegeben. Sollen nur ein paar Spalten ausgewählt werden, so werden diese (im Gegensatz zum SAS Datenschritt) mit einem Komma aufgezählt und dann auch in dieser Reihenfolge ausgegeben. AS weist einer Tabelle oder Spalten einen Alias zu. Vor allem wenn Spaltennamen zwischen Tabellen nicht eindeutig sind, müssen diese mit AS umbenannt werden.

```
PROC SQL;
  VALIDATE
    SELECT *
      FROM sql.mitarbeiter;
QUIT;
NOTE: PROC SQL statement has valid syntax

PROC SQL;
  SELECT vorname,nachname AS familienname,
         (gehalt/12) AS monatsgehalt FORMAT=COMMAX10.2
    FROM sql.mitarbeiter
    ORDER BY monatsgehalt DESC; /*absteigend sortiert*/
QUIT;
```

Tabelle 2: Ergebnis – Beispiel Auswählen von Spalten

vorname	familienname	monatsgehalt
Karla	Humboldt	7.500,00
Jürgen	Bauer	6.733,33
...	...	...

3.2.2 Auswählen von Beobachtungen

Mit WHERE können Beobachtungen, die Bedingungen erfüllen, selektiert werden. Eine Bedingung kann aus mehreren Teilbedingungen, die mit AND (NOT) und OR (NOT) verknüpft werden, bestehen. Eine Teilbedingung besteht aus einem Spaltennamen, einem Operator sowie entweder einer Konstanten, einer weiteren Spalte oder einer Funktion. Als Operatoren stehen die Vergleichsoperatoren (z.B. =, <= etc.) sowie die Operatoren ANY, ALL, BETWEEN AND, CONTAINS, EXISTS, IN, IS NULL oder IS MISSING, LIKE und *= (sounds like) zur Verfügung. Als Funktionen stehen SAS Funktionen zur Auswahl (z.B. MEAN, SUM, ABS, MIN und MAX).

```
PROC SQL;
  SELECT mita_id,vorname,nachname
    FROM sql.mitarbeiter
    WHERE nachname LIKE '%er';
QUIT;
```

Der LIKE Operator ermöglicht eine Suche auf der Grundlage eines Musters mit „%“ und „_“ als Platzhalter. „%“ steht für eine beliebige Anzahl an Zeichen, „_“ steht für ein einziges beliebiges Zeichen. Dieses Beispiel liefert alle Beobachtungen, deren Nachname auf „er“ endet, unabhängig von der Anzahl der vorhergehenden Zeichen.

```
PROC SQL;
  SELECT vorname,nachname,
         (gehalt/12) AS monatsgehalt FORMAT=COMMAX10.2
    FROM sql.mitarbeiter
    WHERE monatsgehalt > 4000;
QUIT;
ERROR: The following columns were not found in the contributing
tables: monatsgehalt.
```

Werden in einer Anfrage neue Spalten erzeugt, so sind diese nicht sofort verfügbar. Eine in einer Anfrage neue erzeugte Spalte ist WHERE nicht bekannt, d.h. es wird die obige Fehlermeldung ausgegeben. Mögliche Lösungen sind:

```
PROC SQL;
     ...
    WHERE (gehalt/12) > 4000;
/*oder*/
     ...
    WHERE CALCULATED monatsgehalt > 4000;
QUIT;
```

Entweder wird die neu angelegte Spalte in der WHERE Anweisung noch einmal spezifiziert oder CALCULATED wird angegeben.

3.2.3 Case Anweisung

Mit der CASE...END Anweisung können neue Spalten erzeugt und Werte, abhängig von Werten bereits bestehender Spalten, zugewiesen werden.

```
PROC SQL;
  SELECT CASE geschlecht
           WHEN 'W' then 'Frau'
           ELSE 'Herr'
         END AS anrede,
         vorname,nachname
    FROM sql.mitarbeiter;
QUIT;
```

Es wird eine neue Spalte anrede erzeugt, die abhängig von Werten in der Spalte geschlecht den Wert „Frau“ oder „Herr“ erhält. Wird kein ELSE Teil angegeben oder trifft kein WHEN Teil zu, werden leere Werte eingetragen.

3.2.4 DISTINCT

DISTINCT liefert nur unterschiedliche Werte einer Variablen oder mehrerer Variablen zurück, d.h. Duplikate werden herausgefiltert.

```
PROC SQL;
  SELECT DISTINCT nachname
    FROM sql.mitarbeiter;
QUIT;
```

Dieses Beispiel liefert alle unterschiedlichen Nachnamen der Mitarbeiter. Zwei Mitarbeiter heißen „Bauer“, durch DISTINCT wird „Bauer“ nur einmal aufgelistet.

3.3 Summenstatistiken mit Aggregatfunktionen

PROC SQL bietet eine Vielzahl von Funktionen mit denen spaltenweise (d.h. über alle Beobachtungen) Kennwerte wie z.B. Summe, Maximum und arithmetisches Mittel berechnet werden können. Bis auf LAG, DIF und SOUND können alle auch im Datenschritt verfügbaren Funktionen verwendet werden. Aggregatfunktionen werden innerhalb der SELECT und HAVING Anweisung spezifiziert. Die Funktion COUNT(var) zählt alle Werte der Variablen var, bei denen die Variable var ungleich MISSING ist. COUNT(DISTINCT var) gibt die Anzahl aller unterschiedlichen, nicht leeren Werte der Variablen zurück.

```
PROC SQL;
  SELECT COUNT(*)
    FROM sql.mitarbeiter;

  SELECT COUNT(DISTINCT nachname)
    FROM sql.mitarbeiter;
QUIT;
```

Die erste Anfrage liefert die Anzahl aller Einträge in der Tabelle sql.mitarbeiter (=15) und die zweite Anfrage die Anzahl aller unterschiedlicher Werte von nachname (=14).

Mit GROUP BY können die Kennwerte auch nur für Untergruppen berechnet werden. Im Gegensatz zum SAS Datenschritt ist es nicht notwendig die Tabelle vorher entsprechend zu sortieren. Ergebnisse aus Aggregatfunktionen können zusätzlich durch HAVING gefiltert werden.

```
PROC SQL;
  SELECT proj_id,COUNT(DISTINCT mita_id) AS anzahl_mitarbeiter
    FROM sql.zuordnung
    GROUP BY proj_id
    HAVING anzahl_mitarbeiter >= 4;
QUIT;
```

Dieses Beispiel liefert alle Projekte zurück, die mindestens 4 (unterschiedliche) Mitarbeiter haben. Erst wird nach proj_id gruppiert, dann die Anzahl aller unterschiedlicher Mitarbeiter innerhalb jeder Gruppe von proj_id gezählt und mit HAVING werden nur die Projekte ausgegeben, die mindestens 4 Mitarbeiter haben.

Hinweis:
Werden neben neu berechneten Kennwerten auch Variablen der Tabelle ausgegeben, werden die Kennwerte in einem zusätzlichen Durchlauf als weitere Variable an jede zurückgegebene Beobachtung angehangen. Es erscheint folgender Hinweis: NOTE: The query requires remerging summary statistics back with the original data.

3.4 Datenschrittoptionen innerhalb von PROC SQL

Tabelle 3 zeigt einige Beispiele wie mit Datenschrittoptionen SQL Anfragen übersichtlicher formuliert bzw. deren Funktionalität erweitert werden kann.

Tabelle 3: Datenschrittoptionen in PROC SQL mit Beispielen

Datenschrittoption	Beispiel	Bemerkungen

LABEL/ FORMAT	`SELECT A LABEL= 'gehalt'`
	`  FORMAT=COMMAX10.`
	`  FROM tab;`
		Benutzerdefinierte und vorhandene Formate können verwendet werden

DROP/KEEP
in FROM TABEL
	`SELECT *`
	`  FROM tab(DROP=A1);`
	
	`SELECT *`
	`  FROM tab(KEEP=A1-A10);`
		Ohne Datenschrittoption müssen alle Spalten einzeln aufgezählt werden

DROP/KEEP
in CREATE
TABLE
	`CREATE TABLE tab (DROP=A1) AS`
	`  SELECT *`
	`  FROM tab1`
	`  WHERE A1=56;`
	
	`/*funktioniert nicht*/`
	`CREATE TABLE tab AS`
	`  SELECT *`
	`  FROM tab1(DROP=A1)`
	`  WHERE A1=56;`
		Zweite Anweisung funktioniert nicht, denn WHERE wird **nach** FROM aber **vor** CREATE TABLE ausgewertet

RENAME
in FROM TABLE
	`SELECT A1 AS B1,…,A9 AS B9`
	`  FROM tab;`
	
	`/*übersichtlicher*/`
	`SELECT *`
	`  FROM tab(RENAME=`
	`  (A1-A9=B1-B9));`
		Ohne Datenschrittoption müssen alle Spalten einzeln umbenannt werden.

WHERE
in FROM TABLE
	`SELECT *`
	`  FROM tab1 INNER JOIN`
	`  (SELECT *`
	`  FROM tab2 WHERE B1<10)`
	`  ON tab1.id=tab2.id;`
		Datenschrittoption WHERE statt einer eigenen SELECT Anfrage

	/*übersichtlicher*/ SELECT * FROM tab1 INNER JOIN tab2(WHERE B1<10) ON tab1.id=tab2.id;	

3.5 Arbeiten mit Views

Views speichern Anfragen und enthalten im Gegensatz zu Tabellen keine Daten. Views sind Dateien, die kompilierten Code enthalten, der erst beim Aufruf der View ausgeführt wird.

```
PROC SQL;
  CREATE VIEW sql.v_mitarbeiter AS
    SELECT *
      FROM sql.mitarbeiter(DROP=gehalt);

    SELECT * /*ausführung der view*/
      FROM sql.v_mitarbeiter;
QUIT;
NOTE: SQL view SQL.V_MITARBEITER has been defined.
```

Mit CREATE VIEW wird die View v_mitarbeiter in der Bibliothek sql erstellt und danach ausgeführt.

Im SAS Explorer Fenster wird ein Symbol für die View angezeigt und eine Datei mit der Endung „sas7bvew“ erstellt. Mit PROC CONTENTS kann man sich Informationen einer View anzeigen lassen. Die Anweisung DESCRIBE VIEW liefert die Anfrage, mit der die View erstellt wurde, und mit DROP VIEW kann eine View auch wieder gelöscht werden.

```
PROC SQL;
  DESCRIBE VIEW sql.v_mitarbeiter;
QUIT;
NOTE: SQL view SQL.V_MITARBEITER is defined as:
select * from SQL.MITARBEITER(DROP=gehalt);
```

Views ermöglichen komplexe Anfragen, sensible oder uninteressante Spalten/Zeilen vor dem Endbenutzer auszublenden (Zugriffskontrolle). Sie eignen sich auch, um Daten aus verschiedenen Datenbasen miteinander zu verknüpfen. Dadurch erlauben Views unterschiedliche Sichten auf gleiche Daten und können somit Datenredundanz vermeiden. Views greifen stets auf die aktuellen Daten der Basistabelle zu. Beispiele von Views sind die SASHELP.Views, die dem Benutzer Zugriff auf die ständig wechselnden Informationen über Bibliotheken, Dateien, Kataloge etc. ermöglichen.

Hinweise:

- Im Vergleich zu Tabellen dauert die Verarbeitung von Views länger, da der Code erst bei Aufruf der View ausgeführt wird.
- Nach Möglichkeit sollte ORDER BY in einer View vermieden werden, denn die Daten werden dann bei jedem Aufruf der View sortiert.

3.6 Verknüpfen von Tabellen

Mit den Join Befehlen können Tabellen ähnlich dem MERGE Befehl horizontal miteinander verknüpft werden. Die Syntax sieht folgendermaßen aus:

```
SELECT ... FROM Tabelle1|View1 <Join Typ> Tabelle2|View2 <ON|WHERE
(Join Bedingung)>
```

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 die Tabellen verknüpft werden sollen. Je nach Join Kriterium kann jeder Join Typ noch zusätzlich als Equi- und Non-Equijoin kategorisiert werden. Equijoins basieren auf einem „=“, Non-Equijoins auf einem „nicht =“ in der Join Bedingung. Folgende Tabellen sollen die Beispiele veranschaulichen. Im Gegensatz zu MERGE im SAS Datenschritt müssen in PROC SQL die Tabellen nicht entsprechend sortiert vorliegen und die Joinspalten können unterschiedliche Namen haben (Ausnahme: Natural Join).

Tabelle 4:sql.gewicht

name	alter	gewicht
Joyce	41	51
John	52	89
Jane	72	70
James	32	87
Alice	43	56

Tabelle 5: sql.groesse

name	alter	groesse
Thomas	85	161
Robert	28	172
Louise	77	152
Jeffrey	34	183
John	52	173

3.6.1 CROSS JOIN

Wird kein Join Kriterium spezifiziert, wird ein CROSS JOIN ausgeführt. Dabei werden alle Zeilen der einen mit allen Zeilen der anderen Tabelle verknüpft. Diese vollständige Kombination wird auch kartesisches Produkt genannt. Aufgrund der immensen Kombinationsmöglichkeiten ist dieser Join sehr rechenintensiv und sollte deshalb vermieden werden.

```
PROC SQL;
  SELECT *
    FROM sql.gewicht AS a,sql.groesse AS b;
  /*oder*/
  SELECT *
    FROM sql.gewicht AS a CROSS JOIN sql.groesse AS b;
QUIT;
```

```
NOTE: The execution of this query involves performing one or more
Cartesian product joins that can not be optimized.
```

Tabelle 6: Ergebnis – CROSS JOIN

name	alter	groesse	name	alter	groesse
Joyce	41	51	Thomas	85	161
Joyce	41	51	Robert	28	172
Joyce	41	51	Louise	77	152
...	...	...	...	...	...

Insgesamt 25 Zeilen (5 Zeilen aus sql.gewicht x 5 Zeilen aus sql.groesse)

Meistens will man nur eine Teilmenge des kartesischen Produktes. Je nachdem welche Teilmenge man möchte, unterscheidet man zwischen INNER und OUTER JOIN.

3.6.2 INNER JOIN

Ein Inner Join gibt nur die Datenreihen aus den Tabellen zurück, welche eine gemeinsame Übereinstimmung nach dem Join Kriterium besitzen. Das Ergebnis eines Inner-Equijoins lässt sich hinsichtlich der spezifizierten Variable als eine Art 'Schnittmenge' der verknüpften Tabellen interpretieren.

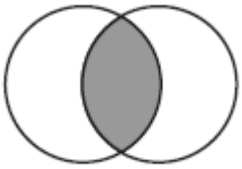

Abbildung 2: INNER JOIN

```
PROC SQL; /*equijoin*/
  SELECT a.*,b.groesse
    FROM sql.gewicht AS a INNER JOIN sql.groesse AS b
    ON a.name = b.name;
QUIT;
```

Tabelle 7: Ergebnis – INNER JOIN

name	alter	gewicht	groesse
John	52	89	173

3.6.3 OUTER JOIN

Outer Joins geben als Ergebnis auch Datenreihen zurück, die keine übereinstimmenden Daten in der anderen Tabelle besitzen. Als Ergebnis wird eine Tabelle ausgegeben, die alle gemeinsamen Beobachtungen enthält und zusätzlich die restlichen Zeilen der linken (LEFT JOIN) Tabelle oder der rechten Tabelle (RIGHT JOIN) bzw. die restlichen Zeilen von beiden Tabellen (FULL JOIN) enthält.

LEFT JOIN

Beim Left Join bzw. Left Outer Join ist die linke Tabelle die Mastertabelle, d.h. alle Beobachtungen der linken Tabelle werden in Kombination mit den übereinstimmenden Beobachtungen der rechten Tabelle ausgegeben. Für Nicht-Übereinstimmungen in der rechten Tabelle werden leere Werte eingetragen.

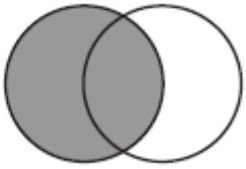

Abbildung 3: LEFT JOIN

```
PROC SQL;
  SELECT a.name as nameA,a.gewicht,
         b.name as nameB,b.groesse
  FROM sql.gewicht AS a LEFT JOIN sql.groesse AS b
  ON a.name = b.name;
QUIT;
```

Tabelle 8: Ergebnis – LEFT JOIN

name	alter	gewicht	groesse
Alice	56		.
James	87		.
Jane	70		.
John	89	John	173
Joyce	51		.

RIGHT JOIN

Beim Right Join bzw. Right Outer Join ist die rechte Tabelle die Mastertabelle, d.h. alle Beobachtungen der rechten Tabelle werden in Kombination mit den übereinstimmenden Beobachtungen der linken Tabelle ausgegeben.

FULL JOIN

Der Full Join bzw. Full Outer Join ist eine Kombination von Left und Right Join. Es werden zusätzlich zu den Datensätzen aus zwei Tabellen, bei denen das Join Kriterium erfüllt ist, auch alle Datensätze aus der linken und der rechten Tabelle mit in das Ergebnis einfügt, die keine Entsprechung in der jeweils anderen Tabelle haben.

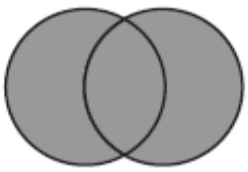

Abbildung 4: FULL JOIN

3.6.4 Weitere Join Typen

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.

NATURAL JOIN

Ein Natural Join verknüpft die Tabellen über die Gleichheit aller jeweils gleichnamigen und gleichtypigen Spalten (d.h. das Join Kriterium ist implizit). Diese gleichnamigen Spalten aus den zwei Tabellen werden im Ergebnis nur einmal angezeigt. Gibt es keine gleichnamigen Spalten, wird der Natural Join zum Cross Join. Mit der Option FEEDBACK, werden die Join Kriterien und der Join Typ im LOG Fenster angezeigt.

```
PROC SQL FEEDBACK;
  SELECT *
    FROM sql.gewicht NATURAL JOIN sql.groesse;
QUIT;
NOTE: Statement transforms to:
select COALESCE(GROESSE.name, GEWICHT.name) as name,
       COALESCE(GROESSE.alter, GEWICHT.alter) as alter,
       GROESSE.groesse, GEWICHT.gewicht
from SQL.GEWICHT inner join SQL.GROESSE on (GROESSE.name =
GEWICHT.name) and (GROESSE.alter= GEWICHT.alter);
```

Die COALESCE Funtion bewirkt hier, dass die gleichnamigen Spalten nur einmal angezeigt werden.

Mehrfach-Join

Da das Ergebnis einer Join Operation wiederum eine Tabelle ist, kann dieses wie eine Tabelle in einer weiteren Join Operation verwendet werden (siehe auch Abbildung 5).

```
PROC SQL; /*mehrfach join*/
  SELECT mita.nachname,mita.vorname,zuor.rolle,
         proj.proj_name,proj.budget
    FROM sql.mitarbeiter AS mita INNER JOIN sql.zuordnung AS zuor
    ON mita.mita_id=zuor.mita_id INNER JOIN sql.projekt AS proj
    ON zuor.proj_id=proj.proj_id
    ORDER BY mita.nachname;
QUIT;
```

sql.zuordnung

mitarbeiter	projekt	mitarbeiter rolle
201	101	SW Entwickler
201	102	SW Entwickler
...	...	...
215	101	Manager
215	102	Manager

sql.mitarbeiter

mitarbeiter	nachname	vorname
201	Betz	Stefan
202	Polak	Edgar
...	...	...
214	Hemmingway	Greg
215	Humboldt	Karla

INNER JOIN

nachname	vorname	mitarbeiter rolle	projekt
Betz	Stefan	SW Entwickler	101
Betz	Stefan	SW Entwickler	102
Betz	Stefan	SW Entwickler	103
...	...	...	...
Hemmingway	Greg	Manager	104
Humboldt	Karla	Manager	101
Humboldt	Karla	Manager	102

sql.projekt

projekt	projekt bezeichnung	gesamtes budget
101	Einführung Data Warehouse	4.000.000
102	Einführung SQL Server	105.000
103	Neues CMS	205.000
...	....	...
106	Neues Corporate Design	30.000

INNER JOIN

nachname	vorname	mitarbeiter rolle	projekt bezeichnung	gesamtes budget
...	...	...	...	...
Betz	Stefan	SW Entwickler	Einführung SQL Server	105.000
Betz	Stefan	SW Entwickler	Einführung Data Warehouse	4.000.000
...	...	...	...	...
Hemmingway	Greg	Manager	Schnittstelle zu Oracle	10.000
Hemmingway	Greg	Manager	Neues CMS	205.000
Humboldt	Karla	Manager	Einführung Data Warehouse	4.000.000
Humboldt	Karla	Manager	Einführung SQL Server	105.000
...	...	...	...	...

Abbildung 5: Ablauf eines Mehrfach-Joins

3.7 Geschachtelte Anfragen (Subqueries)

Subqueries sind SELECT Anweisungen, die selbst eine SELECT Anweisung enthalten. Man spricht von einer „äußeren“ und einer „inneren“ SELECT Anweisung. Die Ergebnisse der inneren Anfrage werden in einer temporären Tabelle gespeichert und dienen als Input für die äußere Anfrage. Die innere Anfrage liefert eine einzige Spalte zurück, mit entweder einem einzigen Wert oder einer Menge von Werten. Im ersten Fall wird

die innere Anfrage durch einen Vergleichsoperator mit der äußeren Anfrage verknüpft. Liefert die innere Anfrage jedoch mehrere Reihen zurück, müssen die Operatoren IN, ALL oder ANY verwendet werden.

```
PROC SQL;
  SELECT *
    FROM sql.zuordnung
    WHERE proj_id IN (SELECT proj_id
                      FROM sql.projekt
                      WHERE proj_name LIKE '%Einführung%');
QUIT;
```

Nach Ausführung der inneren Abfrage, wird die äußere Abfrage zu:

```
PROC SQL;
  SELECT *
    FROM sql.zuordnung
    WHERE proj_id IN (101,102);
QUIT;
```

In den Beispielen bisher war die innere Anfrage von der äußeren Anfrage unabhängig. Man könnte also die innere Anfrage eigenständig ausführen, ihr Ergebnis in eine neue Tabelle eintragen und die äußere Anfrage mit dieser verknüpfen. Jedoch kann die innere Anfrage von der äußeren Anfrage abhängen. In diesem Fall spricht man von einer verzahnt geschachtelten Anfrage oder auch einer korrelierten Subquery. Eine Subquery ist korreliert, wenn sie von den Werten der gerade von der äußeren Anfrage verarbeiteten Zeile abhängig ist. Bei dieser Art von Subquery wird für jede Zeile der äußeren Anfrage die innere Anfrage ausgeführt.

3.8 Verknüpfen von Anfragen (SET-Operatoren)

Mit Hilfe der SET-Operatoren können zwei oder mehr Anfragen, ähnlich der SET Anweisung im Datenschritt, vertikal miteinander verbunden werden. Zu beachten ist, dass die beteiligten Anfragen in Anzahl und Datentyp der selektierten Spalten übereinstimmen müssen und die Verknüpfung standardmäßig entsprechend der Spaltenposition erfolgt. Zwei Beobachtungen werden nur dann als gleich erachtet, wenn sie in allen angegebenen Spalten übereinstimmen. Die Anfragen können vereinigt (OUTER UNION oder UNION), geschnitten (INTERSECT) oder von einander subtrahiert (EXCEPT) werden. UNION, INTERSECT und EXCEPT haben kein direktes Gegenstück im SAS Datenschritt. Die resultierenden Spaltenbezeichungen stammen aus der obersten Anfrage. Mit den beiden Schlüsselwörtern ALL und CORR können die Verknüpfung modifiziert werden.

Tabelle 9: Schlüsselwörter der SET-Operatoren

Schlüsselwort	Erläuterung
ALL	Gleiche Beobachtungen werden nicht unterdrückt. Nicht in Verbindung mit OUTER UNION verwendbar, da hier bereits alle Zeilen ausgegeben werden.
CORR (CORRESPONDING)	Gleichnamige Spalten werden erkannt und überlagert, d.h. die Verknüpfung erfolgt entsprechend der Spaltennamen und nicht nach der Spaltenposition. Bei Verwendung mit UNION, INTERSECT und EXCEPT werden Spalten, die nicht in allen befragten Tabellen vorhanden sind, unterdrückt. In Verbindung mit OUTER UNION werden gemeinsame Spalten überlagert.

3.8.1 UNION

UNION bildet die Vereinigung von zwei Anfragen, indem Zeilen der ersten Anfrage mit allen Zeilen der zweiten Anfrage zusammengefasst werden.

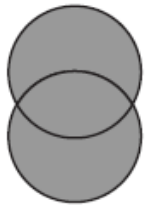

Abbildung 6: UNION Operator

```
PROC SQL;
  SELECT * FROM sql.gewicht
  UNION CORR
  SELECT * FROM sql.groesse;
QUIT;
```

Die Verknüpfung erfolgt entsprechend den gleichnamigen Spalten in beiden Anfragen.

Tabelle 10: Ergebnis – UNION Operator

name	alter
Alice	43
James	32
Jane	72
Jeffrey	34
John	52
Joyce	41
Louise	77
Robert	28
Thomas	85

3.8.2 INTERSECT

Die Ergebnismenge enthält die Schnittmenge der Anfragen, d.h. es werden die Beobachtungen ausgegeben, die in beiden Anfragen enthalten sind.

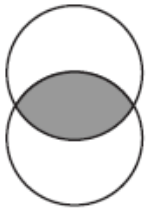

Abbildung 7: INTERSECT Operator

```
PROC SQL;
  SELECT name,alter FROM sql.gewicht
  INTERSECT
  SELECT name,alter FROM sql.groesse;
QUIT;
```

3.8.3 EXCEPT

Die Ergebnismenge enthält die Beobachtungen, die nur in der ersten Anfrage, nicht aber in der zweiten Anfrage enthalten sind.

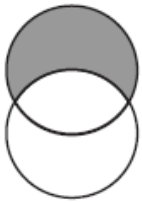

Abbildung 8: EXCEPT Operator

3.8.4 OUTER UNION

Mit OUTER UNION werden Tabellen untereinander gesetzt (konkateniert) und alle Beobachtungen angezeigt. Diese Verknüpfung entspricht SET im Datenschritt.

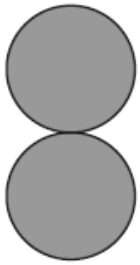

Abbildung 9: OUTER UNION Operator

```
PROC SQL;
  SELECT * FROM sql.gewicht
```

```
  OUTER UNION
  SELECT * FROM sql.groesse;
QUIT;
```

Tabelle 11: Ergebnis – OUTER UNION Operator

name	alter	gewicht	name	alter	groesse
Joyce	41	51		.	.
...	...	...		.	.
Alice	43	56		.	.
	.	.	Thomas	85	161
	.	.	...	...	...
	.	.	John	52	173

4 Weiterführende Themen

In diesem Kapitel werden zwei weiterführende Themen behandelt: die Verbindung von PROC SQL mit der Makrosprache und Indexierung.

4.1 Benutzerdefinierte Makrovariablen und PROC SQL

SAS bietet die Option, PROC SQL mit der Makrosprache zu kombinieren. Das Gute daran ist, man muss kein Experte in der Makroprogrammierung sein, um davon zu profitieren. Mit Hilfe der Erweiterung INTO in der SELECT Klausel können Rückgabewerte an eine Liste von einzelnen Makrovariablen oder an eine einzige Makrovariable übergeben werden. Ein Doppelpunkt (:) wird vor dem Namen der Makrovariablen gesetzt.

```
PROC SQL NOPRINT;
  /*liste von makrovariablen*/
  SELECT vorname INTO :vname1-:vname15
    FROM sql.mitarbeiter;

  /*eine einzige makrovariable*/
  SELECT vorname INTO :vname SEPARATED BY '*'
    FROM sql.mitarbeiter;
QUIT;
%PUT *&vname1* *&vname2* *&vname3* *&vname4* ... *&vname15*;
LOG: *Stefan* *Edgar* *Igor* *Karl* ... *Karla*
%PUT &vname;
LOG: Stefan*Edgar*Igor*Karl*Katrin*Martha*Tobias*Walter*Victor
*Jürgen*Jean*Alberta*William*Greg*Karla
```

4.2 PROC SQL und der Index

Ein Index ist ein Datenbankobjekt, das durch Nutzung einer effektiven Speicherstruktur das Auffinden von Beobachtungen/Datensätzen beschleunigen kann. Prinzipiell ist das Konzept eines Datenbankindexes ähnlich dem eines Buchindexes. Statt eine Tabelle (bzw. das Buch) sequentiell zu durchsuchen, kann durch Suchen im Index der Datensatz schneller ermittelt werden. Die Indexierungsfunktionalität bietet SAS seit Version 6. Obwohl Indizes auch außerhalb von PROC SQL erstellt (z.B. mit Datenschritt oder PROC DATASET) und verwendet werden können, spielen sie vor allem in SQL eine besondere Rolle.

4.2.1 Wie sieht ein Index aus?

Ein Index kann auf einer einzigen Variablen (numerischen oder Character-Variable) oder einer Kombination von Variablen einer Tabelle erstellt werden. Im ersten Fall spricht man von einem simplen Index, sonst von einem zusammengesetzten Index (composite Index). Die indexierten Variablen werden auch Schlüsselvariablen genannt. Es können mehrere Indizes für ein und dieselbe Tabelle erzeugt werden, diese werden in einer mit der Tabelle assoziierten Datei gespeichert (Endung „.sas7bndx“). Der Index speichert aufsteigend die Werte der Schlüsselvariablen und die Adresse der entsprechenden Beobachtungen in der Tabelle. Um eine schnellere und effizientere Suche in einem Index zu ermöglichen, werden Indizes als eine baumartige Datenstruktur angelegt.

4.2.2 Wie wird ein Index angelegt?

```
PROC SQL;
      CREATE UNIQUE INDEX mita_id ON sql.mitarbeiter(mita_id);
      CREATE INDEX ma ON sql.zuordnung(mita_id,proj_id);
      *DROP INDEX mita_id ON sql.mitarbeiter;
QUIT;
```

Es wurden ein simpler und ein composite Index angelegt. Die Option UNIQUE bedeutet, dass die Werte der Schlüsselvariablen eindeutig sein müssen. DROP löscht den Index.

Hinweise:

- Ein simpler Index hat immer den Namen der zugrunde liegenden Schlüsselvariablen, bei einem composite Index wird der Name frei gewählt.
- Sortieren einer indexierten Tabelle nach der Schlüsselvariablen macht den Index ungültig. Deshalb erscheint folgende Fehlermeldung: ERROR: Indexed data set cannot be sorted in place unless the FORCE option is used. Um trotzdem zu sortieren, FORCE Option setzen.

4.2.3 Wann wird der Index verwendet?

SAS kann in folgenden Situationen einen Index zur Optimierung verwenden:

- BY Anweisungen im Datenschritt und Prozedur

- WHERE Anweisungen im Datenschritt und Prozedur
- KEY = im SET und MODIFY DATA im Datenschritt (hier nicht näher erläutert)
- SETKEY in SAS Component Language (hier nicht näher erläutert)
- SQL Join Anweisungen

Die Systemoption MSGLEVEL = I zeigt im SAS LOG Fenster an, ob ein und welcher Index verwendet wurde.

Da ein Index die Werte der Schlüsselvariable(n) aufsteigend sortiert speichert und wiedergibt, kann er für eine BY Anweisung, basierend auf der Schlüsselvariablen, angewendet werden. Damit wird das vorherige Sortieren unnötig. Durch eine WHERE Anweisung wird nur eine Teilmenge von Beobachtungen ausgewählt. Die Verwendung eines Indexes in einer WHERE Anweisungen wird auch „Optimierung der WHERE Anweisung“ genannt. Auch bei einem SQL Join kann SAS entscheiden, einen vorhandenen Index zu nutzen, um das Verknüpfen von Tabellen zu optimieren. Man spricht dann von einem Index Join. Bevor ein Index zur Optimierung verwendet wird, schätzt der SQL Optimierer die Menge, der durch die Anfrage zurückgelieferten Beobachtungen. Wird voraussichtlich eine kleine Teilmenge der gesamten Beobachtungen einer Tabelle zurückgeliefert (1-25%), wird der passende Index zur Optimierung eingesetzt, ansonsten wird das sequentielle Lesen der Tabelle bevorzugt.

4.2.4 Wann sollte ein Index angelegt werden?

Obwohl ein Index die Zugriffszeiten erheblich reduzieren kann, sollte man die Ressourcen (Speicherplatz und CPU Zeit) zur Erstellung und Wartung eines Indexes nicht unbeachtet lassen, denn bei jedem Einfügen, Ändern oder Löschen von Beobachtungen werden alle Indizes automatisch aktualisiert. Deshalb sollte man Folgendes beachten:

- Ist die Tabelle klein (d.h. Anzahl der Speicherseiten < 3. Diese Information erhält man aus PROC CONTENTS) kann sequentielles Lesen schneller sein.
- Wird die Tabelle oft geändert, d.h. Beobachtungen gelöscht/eingefügt, kann der Wartungsaufwand die Vorteile eines Indexes aufheben.
- Liefern die geplanten Anfragen kleine Teilmengen der Gesamtbeobachtungen zurück, so kann ein Index Performanzgewinn bringen.
- Als Schlüsselvariablen eignen sich diskriminierende Variable, auf die oft zugegriffen wird.

5 SAS und relationale Datenbanken (SAS/ACCESS)

Das SAS Modul SAS/ACCESS bietet eine Schnittstelle zwischen der SAS Software und einem relationalen Datenbank Management System (DBMS) und ermöglicht den Zugriff auf dessen Tabellen und Views mittels der LIBNAME Anweisung und der Pass-Through Facility. In den folgenden Anweisungen und Beispielen wurden MySQL Server Version 5.0 und SAS 9.2 mit einem SAS/ACCESS MySQL Treiber (SAS/ACCESS Interface to MYSQL) verwendet.

5.1 LIBNAME Anweisung

In dieser Methode wird mit der LIBNAME Anweisung eine Verbindung zu einer externen Datenbank aufgebaut und diese einer SAS Bibliothek zugewiesen. Das hat den großen Vorteil, dass man nun mit Datenschrittprogrammierung und anderen SAS Prozeduren (auch PROC SQL) auf die Tabellen und Views zugreifen kann. SAS kann viele SAS spezifische Anweisungen in SQL Anweisungen des jeweiligen DBMS übersetzen. Das DBMS führt diese Anweisungen aus und übermittelt die Ergebnisse wieder an SAS. Anweisungen werden optimiert, da sie, wenn möglich, direkt an das DBMS weitergereicht werden und von Indizes der jeweiligen Datenbank Gebrauch machen.

```
OPTIONS SASTRACE=',,,d' SASTRACELOC=SASLOG NOSTSUFFIX;

LIBNAME sqlbib MYSQL USER=&benutzer PASSWORD=&passwort DATABASE=
sqldb SERVER=LOCALHOST PORT=3306;
```

Mit der Systemoption OPTIONS SASTRACE werden die Anweisungen, die an das DBMS geschickt werden, im LOG Fenster angezeigt. Die LIBNAME Anweisung stellt eine Verbindung her zur MySQL Datenbank sqldb und weist dieser eine SAS Bibliothek sqlbib zu. Nun besteht Zugriff auf die MySQL Tabelle class (inhaltsgleich mit der SASHELP.CLASS SAS Datendatei).

```
PROC PRINT DATA=sqlbib.class; RUN;
```

LOG:
```
proc print data=sqlbib.class; run;
     MYSQL_2: Executed: on connection 1
       SELECT * FROM `class`
```

PROC PRINT wird in SQL übersetzt und an das DBMS gesendet. Die Ergebnisse werden von DBMS wieder an SAS übermittelt und angezeigt.

5.2 Pass-Through Facility

Die LIBNAME Verbindung untersützt nur ANSI-SQL. Möchte man die SQL Spracherweiterung der externen Datenbank nutzen, so ist dies nur mit der Pass-Through Facility möglich. Die in SAS formulierten SQL Anweisungen werden direkt an das DBMS zur Ausführung geschickt und können somit z.B. durch die Verwendung eines Indexes optimiert werden. Da mit dieser Methode ausschließlich SQL Anweisungen des jeweilgen DBMS akzeptiert werden, muss der Programmierer mit der SQL Syntax des jeweiligen DBMS vertraut sein.

```
OPTIONS SASTRACE=',,,d' SASTRACELOC=SASLOG NOSTSUFFIX;

PROC SQL;
  CONNECT TO MYSQL AS sqlver (USER=&benutzer PASSWORD=&passwort
  DATABASE=sqldb SERVER=localhost PORT=3306);

  SELECT name,age
    FROM CONNECTION TO sqlver
    (SELECT * FROM class);
```

```
  EXECUTE (CREATE INDEX index_name ON class(NAME)) by sqlver;

  DISCONNECT FROM sqlver;
QUIT;
```

LOG:
```
MYSQL_1: Executed: on connection 0
      SELECT * FROM class
    MYSQL_2: Executed: on connection 0
      CREATE INDEX index_name ON class(NAME)
```

Mit CONNECT TO MYSQL wird eine Verbindung mit dem Alias sqlver zur Datenbank sqldb hergestellt. Die erste Anweisung ist PROC SQL SELECT name,age, die Daten aus der sqlver liest. Die SQL Anweisung, die an das DBMS geschickt wurden, sind nach der CONNECTION TO Komponente spezifiziert (siehe auch Output im LOG Fenster). Mit der EXECUTE Anweisung wird ein Index vom DBMS erzeugt. Mit DISCONNECT FROM wird die Verbindung wieder geschlossen.

Literatur

[1] SAS® 9.1 SQL Procedure User's Guide. Cary, NC: SAS Institute Inc. 2004.

[2] K.P. Lafler: PROC SQL: Beyond the Basics Using SAS®. Cary, NC: SAS Institute Inc. 2004.

[3] K.W. Borowiak: Using Data Set Options in PROC SQL. Proceedings of the 31st Annual SAS Users Group International. 2006.

[4] M.A. Raithel: Creating and Exploiting SAS® Indexes. Proceedings of the 29th Annual SAS Users Group International. 2004.

[5] R. Lavery: The SQL Optimizer Project: _Method and _Tree in SAS®9.1. Proceedings of the 30th Annual SAS Users Group International. 2005.