Proc SQL: Passthrough-Facility für effizientes Datenmanagement bei komplexen großen Datenbeständen

Rainer Kaluscha
Institut für
Rehabilitationsmedizinische
Forschung an der Universität Ulm
Wuhrstr. 2/1
Bad Buchau
rainer.kaluscha@uni-ulm.de

Jakob Holstiege
Institut für
Rehabilitationsmedizinische
Forschung an der Universität Ulm
Wuhrstr. 2/1
Bad Buchau
jakob.holstiege@uni-ulm.de

Gert Krischak
Institut für
Rehabilitationsmedizinische
Forschung an der Universität Ulm
Wuhrstr. 2/1
Bad Buchau
gert.krischak@uni-ulm.de

Zusammenfassung

SAS bietet mit den ACCESS-Modulen via PROC SQL oder LIBNAME-Statements eine flexible und weitgehend datenbankunabhängige Unterstützung für den Zugriff auf relationale Datenbanksysteme. Sind aber in einer komplexen Abfrage viele große Tabellen miteinander zu verbinden (joins) oder soll bereits in der Datenbank hinterlegter Programmcode (stored procedures) genutzt werden, kann es sinnvoll sein, die Pass-Through Facility von PROC SQL zu nutzen, so dass der SQL-Code unverändert an das dahinterliegende Datenbankmanagementsystem durchgereicht und von diesem ausgeführt wird. An einem Beispiel aus der medizinischen Versorgungsforschung werden Vor- und Nachteile dieses Vorgehens beleuchtet.

Schlüsselwörter: PROC SQL, pass through facility, Oracle, stored procedures, Datenmanagement

1 Einleitung

Bei großen Datenbeständen mit komplexen Datenstrukturen ist für ein effizientes Arbeiten ein adäquates Datenmanagement erforderlich. Dabei erlauben ausgereifte relationale Datenbankmanagementsysteme (RDBMS) die performante Abwicklung auch komplexer Abfragen auf großen Datenmengen. Für die Definition der Datenstruktur und von Datenabfragen hat sich das in einem ANSI-Standard normierte SQL (Structured Query Language) etabliert. Allerdings finden sich bei den verschiedenen Datenbank-

systemen gegenüber der im Standard definierten Basissyntax unterschiedliche herstellerspezifische Erweiterungen.
Durch die Bereitstellung des SAS/ACCESS-Moduls unterstützt SAS den Zugriff auf relationale Datenbanksysteme einerseits durch die Zuweisung einer SAS-Bibliothek zur externen Datenbank anhand der Nutzung eines LIBNAME-Statements. Andererseits besteht die Möglichkeit, direkt aus SAS heraus mit der Datenbank in der ihr eigenen Sprache unter Nutzung der SQL Pass-Through-Facility zu kommunizieren.

Auch wenn mit der SAS Version 9.2M3 etliche Verbesserungen bei der impliziten Nutzung von SQL, d.h. der Kommunikation zwischen SAS und RDBMS via ACCESS-Modul, eingeflossen sind [1], gibt es weiterhin sinnvolle Anwendungen der Pass-Through Facility, z.B.:

1. bei Änderungen der Datenbankstruktur (z.B. Neuanlage von Tabellen, Views oder Indizes oder Vergabe von Zugriffsrechten)
2. bei komplexen Verknüpfungen mehrerer großer Tabellen [4]
3. bei Nutzung herstellerspezifischer SQL-Erweiterungen
4. bei Nutzung von in der Datenbank bereits vorhandenem Programmcode (stored procedures)

Dies setzt allerdings eine Vertrautheit mit der Abfragesprache SQL sowie der zugrundeliegenden Datenbank voraus. Ferner muss der SAS-Code bei einem späteren Wechsel des RDBMS ggfs. angepasst werden. Insofern empfiehlt es sich, die datenbankspezifischen Programmteile über zentrale Makros zu realisieren, um den Wartungsaufwand in Grenzen zu halten.

Für alle zur Veranschaulichung gewählten Programmierbeispiele wurde die SAS Version 9.3 und ein Oracle-Datenbanksystem der Version 10g verwendet.

2 Impliziter und expliziter Pass-Through

Das SAS/ACCESS-Modul erlaubt über LIBNAME-Statements, Datenbanktabellen praktisch analog zu gewöhnlichen SAS Datasets anzusprechen.
Beim Zugriff auf den so erstellten Pfad in DATA- oder PROC-Steps übersetzt SAS dies in für die Datenbank verständliche SQL-Anweisungen. Da SAS die verwendete Programmiersyntax selbständig in SQL-Anweisungen für die externe Datenbank übersetzt und an diese übermittelt, wird vom impliziten Durchreichen (implicit SQL pass-through) gesprochen. Mit SAS Version 9.2M3 wurde die Bandbreite der SAS-Befehle, die SAS implizit an das RDBMS übermittelt, weiter vergrößert [1].

Das folgende LIBNAME-Statement illustriert die Zuordnung einer SAS-Bibliothek `REHABIB` zu einer externen ORACLE-Datenbank:

```
LIBNAME REHABIB ORACLE USER=&benutzer PASSWORD=&passwort
   path=&SERVER;

PROC FREQ DATA=REHABIB.TABELLE1;
...
```

Das Schlüsselwort ORACLE weist SAS an, die entsprechende Datenbank-Engine zu nutzen; die notwendigen Parameter wie Benutzername, Passwort und Server folgen. Die konkrete Syntax variiert, da SAS für jedes Datenbanksystem eine eigene Engine nutzt.

Alternativ zum LIBNAME-Statement mit implizitem Pass-Through lassen sich auch SQL-Anweisungen direkt an die Datenbank übermitteln (explizites Pass-Through). Diese Pass-Through Facility ist in PROC SQL eingebettet und wird durch ein CONNECT-Statement eingeleitet:

```
PROC SQL;
     CONNECT to oracle( user=XXX password=YYY
                        path="server/servicename" );
create table work.tabelle1 as
select * from connection to oracle
(
  select * from tabelle1
);
disconnect from oracle;

PROC FREQ DATA=WORK.TABELLE1;
...
```

Einerseits erscheint die Syntax hier gegenüber dem impliziten Pass-Through sperriger, so dass dieser im einfachen Fall vorzuziehen sein dürfte. Anderseits muss sich der Programmierer nicht auf Automatismen verlassen und hat eine feinere Kontrolle über die Abläufe, wenn er den SQL-Code selbst vorgeben kann. Die untenstehenden Beispiele zeigen auf, wo sich dieser Aufwand lohnen kann.

3 Beispiele aus der Praxis

Im Rahmen einer Sekundärdatenanalyse steht ein großer anonymisierter Datensatz zu Teilnehmern medizinischer Rehabilitationsmaßnahmen der Deutschen Rentenversicherung zur Verfügung [2]. Dieser umfasst mehrere Tabellen, deren größte 240 Millionen Datensätze enthält.

3.1 Änderungen der Datenbankstruktur

Hierzu gehört z.B. das Anlegen oder Löschen von Tabellen, Views und Indizes. Untenstehendes Beispiel erzeugt den im Abschnitt 3.2 genutzten Index auf dem Feld Geburtsjahr der Tabelle Versicherte, um effizient nach diesem Merkmal selektieren zu können [3].
Während bei einer einfachen Suche der Zeitbedarf linear mit der Anzahl der Datensätze anwächst, reduziert ein Index die Suchzeit deutlich. So lässt sich etwa in einem alphabetisch sortierten Telefonbuch schnell nach Familie Müller suchen. Dies entspricht einer Datenbanktabelle mit einem Index auf dem Feld Nachname.
Bei Feldern ohne Index ist hingegen eine aufwändige Suche erforderlich: möchte man alle Bewohner der Lindenstraße finden, muss das komplette Telefonbuch von vorne bis hinten durchsucht werden.

Da der SQL-Code nur zum Anlegen des Index dient und keine Rückgabedaten für ein Dataset erwartet werden, wird der SQL-Code in eine EXECUTE-Klausel eingebettet:

```
PROC SQL feedback;
CONNECT to oracle( user=X password=Y
                   path="server.intern/oracle.server.intern" );

EXECUTE( create index I_GBJA on versicherte( gbja ) )
by oracle;
DISCONNECT from oracle;
```

Wenn keine weiteren Datenbankzugriffe mehr erfolgen sollen, empfiehlt es sich, abschließend die Datenbankverbindung mit einem DISCONNECT-Statement abzubauen.

Die FEEDBACK-Option veranlasst SAS, evtl. von der Datenbank gelieferte Meldungen auszugeben. Führt man den Beispielcode zum Anlegen des Index etwa zum zweiten Mal aus, erhält man, weil der Index bereits existiert, die nachfolgende Oracle-Fehlermeldung:

```
ERROR: ORACLE Execute error: ORA-01408: Diese Spaltenliste hat
bereits einen Index.
```

3.2 Komplexe Verknüpfungen mehrerer großer Tabellen

Zwei oder mehr sehr große Tabellen sollen miteinander verknüpft und nach einer komplexen Bedingung gefiltert werden. Die RDBMS verfügen in der Regel über ausgefeilte Strategien zur Anfrageoptimierung (Query Optimizer) und können daher auch solche komplexen Anfragen effizient bearbeiten. Dennoch kann es vorkommen, dass der Mensch über inhaltliches Zusatzwissen verfügt und er daher effizientere Zugriffspfade als der Query Optimizer angeben kann.
Im untenstehenden Beispiel soll für den Geburtsjahrgang 1957 die Art der Beitragsmonate analysiert werden. Da der Programmierer weiß, dass der Geburtsjahrgang als Kri-

terium eine hohe Selektivität aufweist und das Feld indexiert ist, übergibt er an die Oracle-Datenbank einen herstellerspezifischen Hinweis (Optimizer Hint), dass die Anfrage bevorzugt unter Nutzung dieses Indexes abzuarbeiten ist. Da dieser Hinweis laut Oracle-Syntax in einen SQL-Kommentar einzubetten ist, muss die Option PRESERVE_COMMENTS von PROC SQL aktiviert werden. Ansonsten würde PROC SQL diesen Kommentar ausfiltern, bevor der SQL-Code an das RDBMS durchgereicht wird. So kann der Mensch sein Zusatzwissen über Strukturen in den Daten einbringen und die Abarbeitung der Anfrage feiner steuern, falls die Automatismen von SAS und/oder dem RDBMS keine optimalen Zugriffsstrategien ergeben.

Ferner erzwingt der explizite Pass-Through, dass die Verknüpfung der Tabellen auf dem leistungsfähigen Datenbankserver erfolgt und nur eine relativ kleine Datenmenge als Ergebnis über Netzwerk übertragen werden muss. Sonst könnte es passieren, dass PROC SQL beide Tabellen komplett auf den Client-PC kopiert und die Verknüpfung dort selbst durchführt, da komplexe Verknüpfungen die Nutzung des impliziten Pass-Through unterbinden können [1]. Dies belastet dann Server, Client und Netzwerk unnötig. Da i.d.R. die Hardware des Client-PC der des Servers unterlegen ist, führt dies auch zu unnötig langen Laufzeiten oder Fehlermeldungen, z.B. aufgrund von Speichermangel. Der hier verwendete Datenbankserver ist hingegen mit 256 GB RAM so ausgelegt, dass auch größere Datensätze komplett im Hauptspeicher gehalten werden können und somit keine (vergleichsweise langsamen) Zugriffe auf Festplatten erforderlich werden.

```
proc sql;
 connect to oracle( user=X password=Y
                    path="server.intern/oracle.server.intern"
                    PRESERVE_COMMENTS );

create table work.test as select * from connection to oracle
(
select /*+INDEX(v I_GBJA) */ *
from versicherte v, beitragsmonate b
where v.case = b.case and v.gbja = 1957 and ...
);
disconnect from oracle;
```

3.3 Nutzung herstellerspezifischer SQL-Erweiterungen

Herstellerspezifische Erweiterungen können nur unter Anwendung der Pass-Through Facility aus SAS heraus genutzt werden [3]. So bietet Oracle z.B. den SOUNDEX-Operator an, der die Suche nach ähnlich klingenden Wörtern ermöglicht. Ferner gibt es viele nützliche Funktionen, um etwa Datumswerte zu manipulieren (z.B. „Rundung“ auf den Monatsersten) oder in Zeichenketten zu wandeln.

3.4 Nutzung von in der Datenbank bereits vorhandenem Programmcode (stored procedures)

Für die Realisierung eigener Funktionen bietet Oracle die Programmiersprache PL/SQL an, die neben SQL-Abfragen auch die üblichen Konstrukte imperativer Programmiersprachen wie bedingte Verzweigungen, Schleifen oder Funktionsaufrufe bereitstellt [5].

Für einen Überblick über Erwerbsverläufe ist es wichtig, Lücken bei den Beitragszahlungen zu erkennen. In einer Tabelle liegen jahresweise das erzielte sozialversicherungspflichtige Entgelt und die daraus gezahlten Beiträge vor.

Die unten dargestellte PL/SQL-Funktion visualisiert die jahresweisen Beiträge eines Versicherten als Zeichenkette (String), indem für Jahre mit sozialversicherungspflichtigem Entgelt, d.h. Beitragszahlungen, ein ‚X' und für Jahre ohne Beiträge ein Leerzeichen eingetragen wird. Dazu werden die Datensätze eines über die Fallnummer (Parameter FALL) identifizierten Versicherten innerhalb eines vorgegebenen Zeitfensters (Parameter VON und BIS) selektiert und nach dem Jahr der Beitragszahlung (Feld BEIJAHR) sortiert. Für evtl. Lücken bis zum nächsten Beitrag wird dabei mittels der RPAD-Funktion eine entsprechende Anzahl Leerzeichen angehängt.

Da auf dem Feld Fallnummer (CASE) der Beitragstabelle ein Index liegt, können die Beiträge zu einem gegebenen Fall rasch gefunden werden. Dadurch arbeitet die Funktion sehr performant, obwohl die Tabelle zwanzig Millionen Datensätze enthält.

Der untenstehende Codeblock beinhaltet den zum Anlegen der Funktion nötigen PL/SQL-Code. Dieser kann mit Oracle-Tools wie etwa dem SQL-Developer [6] oder innerhalb einer EXECUTE-Klausel (vgl. 3.1) von PROC SQL ausgeführt werden:

```
CREATE or replace FUNCTION beitragsmuster(
      fall  INTEGER,
      von   INTEGER DEFAULT 1999,
      bis   INTEGER DEFAULT 2009 )
RETURN VARCHAR2
IS
    s   VARCHAR2(16) := ' ';
BEGIN

    for c in ( select beijahr from beitraege
               where case = fall
                 and beijahr between von and bis
                order by beijahr )
    loop
        s  := rpad( s, c.beijahr - von + 1, '_' ) || 'X';
    end loop;
```

```
    return rpad( s, bis - von + 2, '_' );
END;
/
show errors
```

Nun kann die neu geschaffene PL/SQL-Funktion BEITRAGSMUSTER in Datenbankabfragen genutzt werden, um etwa für den Geburtsjahrgang 1957 die Beitragsverläufe zu generieren und auszuzählen:

```
PROC SQL;
 connect to oracle( user=X password=Y
                    path="server.intern/oracle.server.intern");

create table work.test as select * from connection to oracle
(
  select v.case, beitragsmuster( v.case ) beitragsmuster
  from versicherte v
  where v.gbja = 1957 and ...
);
disconnect from oracle;

PROC FREQ data= work.test order=freq;
  TABLES beitragsmuster;
run;
```

Abbildung 1: Häufigkeit der Beitragsmuster

Die Prozedur FREQ

BEITRAGSMUSTER

BEITRAGSMUSTER	Häufigkeit	Prozent	Kumulative Häufigkeit	Kumulativer Prozentwert
xxxxxxxxxxx	139534	51.36	139534	51.36
_____	73096	26.91	212630	78.27
xxxxxxxxxx_	4149	1.53	216779	79.80
xxxxxxxxx__	3492	1.29	220271	81.08
xxxxxxxx___	2628	0.97	222899	82.05
x_____	2577	0.95	225476	83.00
...	...	...	...	...

Im SELECT-Kommando muss der Funktionsaufruf einen Namen (Alias, hier: beitragsmuster) erhalten, den SAS dann im resultierenden Dataset als Variablennamen übernimmt. Die Erzeugung der Beitragsmuster aus elf Jahren für 270.000 Versicherte benötigt dann lediglich eine halbe Minute (s. Abbildung 1).

So lässt sich schnell erkennen, dass die häufigsten Varianten durchgängige Beitragszahlungen oder keine Beitragszahlungen (etwa bei latent Versicherten) sind, während Beitragszahlungen mit Unterbrechungen relativ selten auftreten.

5 Zusammenfassung

Die implizite Nutzung von SQL hat sich bei SAS stetig weiterentwickelt [1], so dass bei PROC SQL und LIBNAME-Statements mittlerweile vieles automatisch ablaufen kann.
Dennoch gibt es Situationen, wo die explizite Programmierung in SQL mit anschließendem Durchreichen des Codes an das Datenbanksystem (explicit Pass-Through) ihre Berechtigung behält.
Beide Vorgehensweisen lassen sich in SAS flexibel mischen, so dass abhängig vom Zweck und Aufwand die effizienteste Variante gewählt werden kann.

Literatur

[1] Capobianco, F: Explicit SQL Pass-Through: Is It Still Useful?
SAS Global Forum, 2011.
Online: http://support.sas.com/resources/papers/proceedings11/105-2011.pdf

[2] Forschungsdatenzentrum der Rentenversicherung: „Scientific Use File: Abgeschlossene Rehabilitation im Versicherungsverlauf 2002 – 2009 (SUFRSDLV09B)“. Online: http://www.fdz-rv.de

[3] Borowiak , KW: Effectively Using the Indices in an Oracle Database with SAS.
North East SAS Users Group, 2004.
Online: http://www.nesug.org/proceedings/nesug04/po/po12.pdf

[4] Weires, M (2009). SQL Processing mit der SAS Software: Eine Einführung in die Prozedur SQL.
13. Konferenz der SAS-Anwender in Forschung und Entwicklung (KSFE), 2009.
Online: http://de.saswiki.org/images/8/8a/13.KSFE-2009-Weires-Einfuehrung_in_SQL.pdf

[5] Oracle Technology Network: Database Features - PL/SQL.
Online: http://www.oracle.com/technetwork/database/features/plsql/

[6] Oracle Technology Network: Developer Tools - SQL Developer.
Online: http://www.oracle.com/technetwork/developer-tools/sql-developer/