BISSANTZ

Der SQL-Durchgriff 2.0

Autor	Benjamin Schwegler
Datum der Veröffentlichung	01.02.2019

Bissantz & Company GmbH
Nordring 98 90409 Nürnberg
T +49 911 935536 - 0
F +49 911 935536 - 10
service@bissantz.de

Geschäftsführer
Dr. Nicolas Bissantz
Michael Westphal

USt-IdNr. DE181064258
Steuer-Nr. 241/118/25048
Amtsgericht Nürnberg
HRB 15590

Sparkasse Nürnberg
IBAN DE88 7605 0101 0570 1792 91
BIC SSKNDE77XXX

www.bissantz.de

Abstract

Der folgende Beitrag soll die grundsätzlichen Funktionen des seit DeltaMaster 6.2.1 existierenden SQL-Durchgriffs 2.0 auf Basis von Prozeduren erläutern. Es wird auf die Schritte, die nötig sind, um diese Art des SQL-Durchgriffs zu verwenden, die Struktur der nötigen Prozeduren sowie die Vor- und Nachteile des neuen Moduls eingegangen. Zwei grundsätzliche Implementierungsmöglichkeiten der Prozeduren werden aufgezeigt, sowie eine Möglichkeit, generische Prozeduren für Modelle, die auf DeltaMaster ETL basieren, zu erzeugen.

Der SQL-Durchgriff 2.0

Einleitung

DeltaMaster bietet mit dem SQL-Durchgriff die Möglichkeit, durch Anbindung des relationalen Modells vom OLAP-Cube mit voraggregierten Daten auf die zugrundeliegenden Bewegungsdaten der Faktentabelle zuzugreifen, falls es nötig sein sollte, Details abzurufen, die nicht in die OLAP-Modellierung aufgenommen wurden. Eine Modellierung in DeltaMaster ETL als MeasureGroup Info schreibt solche Details in die Faktentabellen und macht diese damit direkt abrufbar mithilfe des SQL-Durchgriffs. Für Erweiterungen wie Berechtigungen, Partitionen oder berechnete Kennzahlen gibt es zusätzlich die Möglichkeit der Anbindung einer V_SEC-View (Informationen hierzu siehe Beitrag vom 1.7.2011 „DeltaMaster SQL-Durchgriff optimieren“). Seit Version 6.2.1 erlaubt DeltaMaster nun neben der Anbindung einer Tabelle bzw. einer View auch die Anbindung einer Prozedur. Die neuen Funktionalitäten werden im Folgenden dargestellt.

Vorbereitung

Um den SQL-Durchgriff 2.0 verwenden zu können, muss in der jeweiligen Anwendung zunächst das relationale Modell angebunden. Außerdem muss in der Konfigurationsdatei DeltaMaster.exe.config der Key „EnableSQLDT2Reports“ auf „true“ gesetzt werden:

DeltaMaster.exe.config

```
1  <?xml version="1.0"?>
2  <configuration>
3    <appSettings>
4      <add key="EnableSQLDT2Reports" value="true"/>
5    </appSettings>
6      <runtime>
7          <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
```

Abbildung 1: Eintrag in DeltaMaster.exe.config

Im Anschluss daran muss DeltaMaster neu gestartet und die Anwendung per Rechtsklick geöffnet werden. Falls bereits SQL-Durchgriffe in der Anwendung erstellt wurden, muss die Anwendung zunächst über den Optionsdialog geöffnet werden. Im erscheinenden Dialog wird „Berichte in neues Format konvertieren“ ausgewählt:

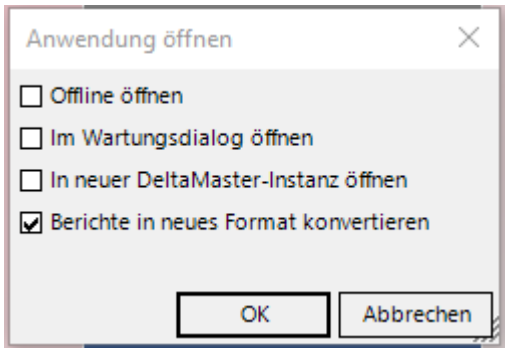

Abbildung 2: Optionsdialog

Jetzt sind in der Anwendung auch die bestehenden SQL-Durchgriffe Berichte des neuen Moduls. Diese auf Tabellen basierenden SQL-Durchgriffe behalten dabei ihre volle Funktionalität.

Dieser Schritt ist für jede Anwendung nach Abspeichern genau einmal notwendig.

Nachdem die Anwendung auf diese Weise geöffnet wurde, haben die Einstellungen des bereits erstellten SQL-Durchgriffs im Reiter „Allgemein“ eine zusätzliche Auswahlmöglichkeit „Daten-Durchgriff auf Basis“:

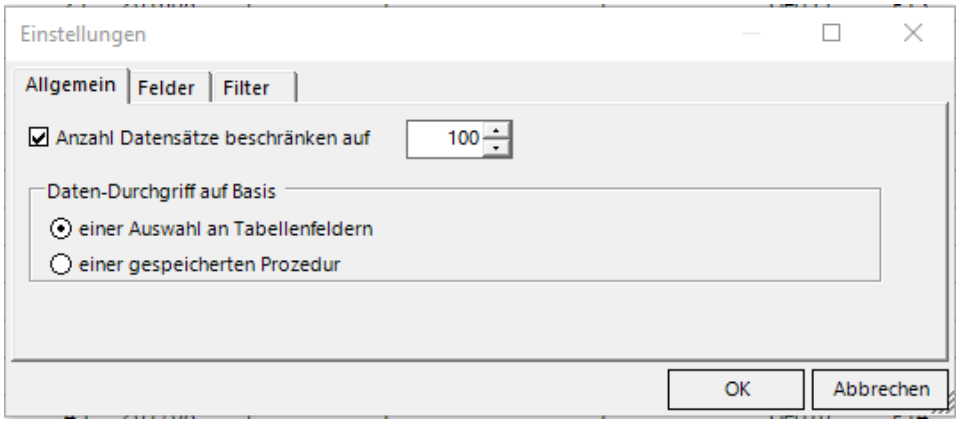

Abbildung 3: Einstellungen des SQL-Durchgriffs 2.0

Die Einstellung „einer Auswahl an Tabellenfeldern“ führt zu dem SQL-Durchgriff, wie er aus DeltaMaster bislang bekannt ist. Die Auswahl „einer gespeicherten Prozedur“ führt zu dem neuen SQL-Durchgriff. Die manuelle Eingabe des Namens einer kompatiblen Prozedur führt zu einer ähnlichen Eingabemaske wie beim gewöhnlichen SQL-Durchgriff:

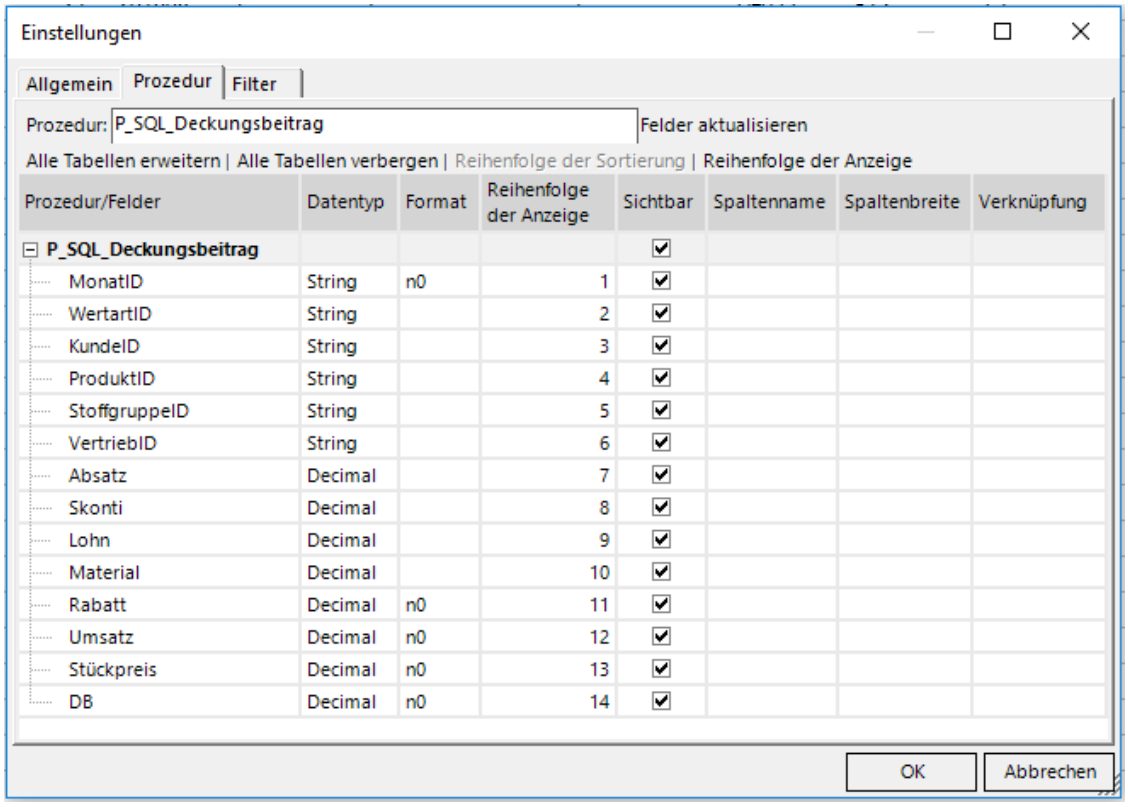

Abbildung 4: SQL-Durchgriff Metadaten

Struktur der Prozedur

DeltaMaster erwartet einen Parameter *@maxRows int*, der immer mitgegeben werden muss. Zum Erstellen der Einstellungsmaske wird bei Klicken von „Felder aktualisieren" die Prozedur ohne Parameter außer *@maxRows* = 0 aufgerufen, um die Metadaten der Prozedur zu erhalten. Es ist daher aus Performancegründen ggf. sinnvoll, diesen Fall abzufangen und die Metadaten ohne weitere Berechnungen an DeltaMaster zu übergeben.

Des Weiteren werden die Filtereinstellungen der View als Parameter weitergegeben, falls in der jeweiligen Hierarchie ein Element, das vom All-Element verschieden ist, ausgewählt ist. Seit DeltaMaster 6.2.3 werden die ID-Spalten mit ihrer Bezeichnung weitergegeben, an Stelle der Bezeichnungen der Aliasspalten. Alle Hierarchieebenen, die filterbar sein sollen, müssen der Prozedur als Parameter bereitgestellt werden. Eine einfache Möglichkeit, einen Überblick über die Parameter zu bekommen, ist es, die Prozedur lediglich mit dem *@maxRows* Parameter auszustatten, danach in jeder Dimension ein Element auszuwählen und im SQL-Log anzusehen, welche Parameter erwartet werden:

EXECUTE P_SQL_Deckungsbeitrag

@MaxRows = 100,

@EinheitID = N'1',

@KumulationID = N'1',

@RegionID = N'1',

@JahrID = N'2018',

@PeriodenansichtID = N'temp',

@ProdukthauptgruppeID = N'1',

@StoffgruppeID = N'C1',

@VertriebID = N'V1',

@WertartID = N'I'

->

Für die Prozedur oder Funktion P_SQL_Deckungsbeitrag wurden zu viele Argumente angegeben.

Es ist zu erkennen, dass alle Parameter als Unicode-Zeichenfolge weitergegeben werden. Wird ein Parameter in einem anderen Format benötigt, kann dies problemlos durch implizite oder explizite Typkonvertierungen durchgeführt werden.

Das Resultat, das im DeltaMaster angezeigt wird, ist immer die erste SQL-Query, die in der Prozedur ausgegeben wird. Die Prozedur sollte daher immer mindestens ein Select-Statement enthalten, das als Ergebnis dient. Der Inhalt dieses Select-Statements ist jedoch genau wie der restliche Inhalt der Prozedur völlig frei vom Benutzer definierbar.

Implementierung

Im Folgenden wird anhand der Chair-Datenbank gezeigt, welche Möglichkeiten der SQL-Durchgriff mithilfe von Prozeduren bietet. Zunächst wird mithilfe der Kundendimension gezeigt, wie eine Prozedur für den SQL-Durchgriff prinzipiell aussehen kann.

```
CREATE Procedure [dbo].[P_SQL_Deckungsbeitrag]
(
@MaxRows int = 0,
@EinheitID nvarchar(max) = N'1',
@KumulationID nvarchar(max) = N'1',
@PeriodenansichtID nvarchar(max) = N'1',
@WertartID nvarchar(max) = N'I',
@RegionID nvarchar(max) = NULL,
@GebietID nvarchar(max) = NULL,
@PLZID nvarchar(max) = NULL,
@KundeID nvarchar(max) = NULL,
@VertriebskanalID nvarchar(max) = NULL,
@BrancheID nvarchar(max) = NULL,
@JahrID nvarchar(max) = NULL,
@QuartalID nvarchar(max) = NULL,
@MonatID nvarchar(max) = NULL,
@ProdukthauptgruppeID nvarchar(max) = NULL,
@ProduktgruppeID nvarchar(max) = NULL,
@ProduktID nvarchar(max) = NULL,
@StoffgruppeID nvarchar(max) = NULL,
@VertriebID nvarchar(max) = NULL
)
AS
SELECT TOP (SELECT @MaxRows)            -- optional, wird benötigt, um die Anzahl der an-
gezeigten Zeilen im DeltaMaster steuern zu können
       k4.KundeBEZ AS [Kunde]
     ,[Absatz]
     ,[Umsatz]
     ,Umsatz/NULLIF(Absatz,0) AS [Stückpreis]
     ,Umsatz - Skonti - Lohn - Material - Rabatt AS [DB]
FROM [Chair].[dbo].[T_FACT_01_Deckungsbeitragsrechnung] fact
LEFT JOIN T_DIM_05_04_Kunde k4
       ON fact.KundeID=k4.KundeID
LEFT JOIN T_DIM_05_03_PLZ k3
       ON k3.PLZID=k4.PLZID
LEFT JOIN T_DIM_05_02_Gebiet k2
       ON k2.GebietID=k3.GebietID
LEFT JOIN T_DIM_05_01_Region k1
       ON k1.RegionID=k2.RegionID
WHERE @maxrows>0
AND ISNULL(k1.RegionID,-1)=ISNULL(@RegionID,ISNULL(k1.RegionID,-1))
AND ISNULL(k2.GebietID,-1)=ISNULL(@GebietID,ISNULL(k2.GebietID,-1))
AND ISNULL(k3.PLZID,-1)=ISNULL(@PLZID,ISNULL(k3.PLZID,-1))
AND ISNULL(k4.KundeID,-1)=ISNULL(@KundeID,ISNULL(k4.KundeID,-1))
AND ISNULL(k4.BrancheID,-1)=ISNULL(@BrancheID,ISNULL(k4.BrancheID,-1))

AND   ISNULL(k4.VertriebskanalID,-1)=ISNULL(@VertriebskanalID,ISNULL(k4.VertriebskanalID,-
1))
```

Hier wurde lediglich die Kundendimension im DeltaMaster filterbar gemacht. Für jede auswählbare Ebene wird, falls ein Element übergeben wird, auf dieses gefiltert. Falls ein Parameter den Wert NULL hat, wird sichergestellt, dass das die jeweilige Ebene betreffende Statement immer TRUE ergibt und somit in diesem Falle keine Filterung durchgeführt wird. Es wird deutlich, dass diese Prozeduren schnell sehr aufwändig werden können. Des Weiteren ist diese Implementierung mit den Joins über alle Dimensionsebenen sehr unflexibel. Diese Implementierung hat gegenüber den V_SEC_Views den Vorteil, dass Filterungen im DeltaMaster sowohl ignoriert als auch zu beliebigen anderen Zwecken verwendet werden können.

Weitere Flexibilität bietet die Implementierung mithilfe von dynamischem SQL:

```
CREATE Procedure [dbo].[P_SQL_Deckungsbeitrag_Dynamisch]
(
@MaxRows int = 0,
@EinheitID nvarchar(max) = N'1',
@KumulationID nvarchar(max) = N'1',
@PeriodenansichtID nvarchar(max) = N'1',
@WertartID nvarchar(max) = N'I',
@RegionID nvarchar(max) = NULL,
@GebietID nvarchar(max) = NULL,
@PLZID nvarchar(max) = NULL,
@KundeID nvarchar(max) = NULL,
@VertriebskanalID nvarchar(max) = NULL,
@BrancheID nvarchar(max) = NULL,
@JahrID nvarchar(max) = NULL,
@QuartalID nvarchar(max) = NULL,
@MonatID nvarchar(max) = NULL,
@ProdukthauptgruppeID nvarchar(max) = NULL,
@ProduktgruppeID nvarchar(max) = NULL,
@ProduktID nvarchar(max) = NULL,
@StoffgruppeID nvarchar(max) = NULL,
@VertriebID nvarchar(max) = NULL
)
AS
DECLARE @SQL varchar(max)
SET @SQL =
'SELECT TOP '+CONVERT(varchar,@MaxRows)+'
       '+IIF(COALESCE(@KundeID,@PLZID,@GebietID,@RegionID,@BrancheID,@VertriebskanalID) IS
NULL,'fact.KundeID','k4.KundeBEZ')+' AS [Kunde]
     ,[Absatz]
     ,[Umsatz]
     ,Umsatz/NULLIF(Absatz,0) AS [Stückpreis]
     ,Umsatz - Skonti - Lohn - Material - Rabatt AS [DB]
FROM [Chair].[dbo].[T_FACT_01_Deckungsbeitragsrechnung] fact
'+IIF(COALESCE(@KundeID,@PLZID,@GebietID,@RegionID,@BrancheID,@VertriebskanalID)       IS
NULL,'','LEFT JOIN T_DIM_05_04_Kunde k4 ON fact.KundeID=k4.KundeID
')+IIF(COALESCE(@PLZID,@GebietID,@RegionID) IS NULL,'','LEFT JOIN T_DIM_05_03_PLZ k3 ON
k3.PLZID=k4.PLZID
')+IIF(COALESCE(@GebietID,@RegionID) IS NULL,'','LEFT JOIN T_DIM_05_02_Gebiet k2 ON k2.Ge-
bietID=k3.GebietID
')+IIF(@RegionID IS NULL,'','LEFT JOIN T_DIM_05_01_Region k1 ON k1.RegionID=k2.RegionID
')+'WHERE '+CONVERT(varchar,@MaxRows)+'>0
'+IIF(@RegionID IS NULL,'','AND k1.RegionID='''+@RegionID+'''
')+IIF(@GebietID IS NULL,'','AND k2.GebietID='''+@GebietID+'''
')+IIF(@PLZID IS NULL,'','AND k3.PLZID='''+@PLZID+'''
')+IIF(@KundeID IS NULL,'','AND k4.KundeID='''+@KundeID+'''
')+IIF(@BrancheID IS NULL,'','AND k4.BrancheID='''+@BrancheID+'''
')+IIF(@VertriebskanalID IS NULL,'','AND k4.VertriebskanalID='''+@VertriebskanalID+'''
')

exec (@sql)
```

Diese Implementierung bietet den zusätzlichen Vorteil, dass Joins, sowie Where Bedingungen oder Order-by-Statements beliebig gesetzt werden können. Es ist zum Beispiel möglich, mithilfe einer Dummy-Dimension in DeltaMaster zwischen benutzerdefinierten Sortierungen zu wechseln. Ferner ist mit dynamischem SQL nicht nur der Inhalt, sondern auch der Name einer Spalte flexibel wählbar.

Der SQL-Durchgriff 2.0. ermöglicht außerdem die Übergabe von berechneten Elementen, wenn bei der Implementierung die DeltaMaster-Analysesitzung und die SQL-Prozedur aufeinander abgestimmt sind. Hierfür ist es ausreichend, die ID des berechneten Elementes ([tempX]) zu kennen und in der Prozedur zu berücksichtigen.

Des Weiteren ist es möglich, Transaktionen außerhalb eines Select-Statements auszuführen. So kann eine Abfrage, die in einer V_SEC_View mithilfe von CTEs implementiert wurde, nun in eine Prozedur mit temporären Tabellen geschrieben werden, um eine Performance-Verbesserung zu erreichen. Des Weiteren ist es möglich, ein Logging zu definieren, um herauszufinden, wann, wie oft und mit welchen Parametern der Bericht geöffnet wird.

Nachteile

Bei Implementierung eines SQL-Durchgriffs mit Prozeduren muss man sich bewusst sein, dass bei Aufruf des Berichtes der User eine Prozedur startet. Das bedeutet, dass die Berechtigungen für diese Prozedur sehr sorgfältig gepflegt werden müssen.

Im Gegensatz zum bekannten SQL-Durchgriff unterstützt der SQL-Durchgriff 2.0 mit Prozeduren keine Mehrfachauswahl in den Filtereinstellungen. Es ist zwar möglich, dies mit berechneten Elementen zu umgehen, allerdings kann dies bei Anwendern für Verwirrung sorgen.

Eine Implementierung der SQL-Prozeduren ist sehr zeitaufwendig. Diesem Beitrag ist eine Prozedur P_Create_SQL_Proc beigefügt, die für Modelle, die mit DeltaMaster ETL generiert wurden, durch simplen Aufruf der Prozedur mit der ID der gewünschten Measuregroup eine Prozedur generiert, die so gut wie alle Funktionalitäten des SQL-Durchgriffs auf Basis der Faktentabelle beinhaltet. Lediglich DeltaMaster-spezifische berechnete Elemente können von der Prozedur, im Gegensatz zu SQL-Durchgriffen auf Basis von Faktentabellen, nicht automatisch verwendet werden. Die durch die Prozedur P_Create_SQL_Proc erzeugten Prozeduren sind jedoch lediglich als Starthilfe gedacht. Weitere projektspezifische Anwendungen müssen im Anschluss immer noch durchgeführt werden. Es sollte also im Einzelnen überdacht werden, ob die zusätzlichen Funktionalitäten den Aufwand rechtfertigen.

Fazit

Der SQL-Durchgriff 2.0 erweitert die Vielseitigkeit des SQL-Durchgriffs ungemein. Wenn sehr spezielle Abfragen bezüglich Komplexität oder Performance gewünscht werden, die bisher mit DeltaMaster nicht abbildbar waren, kann der SQL-Durchgriff auf Basis von Prozeduren Abhilfe schaffen. Allerdings ist hierbei Vorsicht geboten, denn gegebenenfalls steht der größere Aufwand nicht im Verhältnis zu den zusätzlichen Funktionen. Es sollte daher von Fall zu Fall abgewogen werden, ob eine Implementierung einer V_SEC_View nicht ausreichend ist. Ideal ist der SQL-Durchgriff auf Basis von Prozeduren als Absprungbericht zu verwenden, da hier durch benutzerdefinierte Analysepfade in der Anwendung die Eingabeparameter der Prozedur so eingeschränkt werden können, dass sich die Implementierung vereinfacht und das Ergebnis genau auf die jeweilige Fragestellung angepasst werden kann.