BISSANTZ

Magischer SQL-Durchgriff

Autor	Tobias Krebber
Datum der Veröffentlichung	16.11.2018

Abstract

Der eingabefähige DeltaMaster-SQL-Durchgriff ist ein wirksames Instrument zur relationalen Datenerfassung. Er bietet mit Auswahlprozeduren, Spaltenreferenzen und Defaultwerten darüber hinaus noch einige Möglichkeiten, die den Eingabeprozess dynamischer gestalten. Zusätzlich kann durch die Verwendung von Views eine Aggregation oder die Ermittlung von Vorjahreswerten vorgenommen werden.

Magischer SQL-Durchgriff

1 Einleitung

DeltaMaster bietet mit dem eingabefähigen SQL-Durchgriff ein praktisches Werkzeug um relationale Tabellen zu pflegen. Deshalb ist dieses Feature besonders bei der Stammdatenpflege sehr beliebt. Doch auch bei der Planung kann es Situation geben, in denen die zu beplanenden Strukturen erst durch die Planung entstehen. Hier kann der SQL-Durchgriff Abhilfe schaffen. Dieser Schritt bedeutet aber immer das Verzichten auf gewisse, aus der DeltaMaster-Planung bekannte, Funktionen. Dank Default-Auswahlen und Auswahlprozeduren für Felder ist die Situation allerdings nicht so aussichtslos, wie man zunächst annehmen mag. Dieser Blog gibt einen Überblick, wie man sich auch relational helfen kann.

2 Ausgangslage

Zur Demonstration wird ein einfaches Datenmodell herangezogen. Zentrale Komponente ist die Tabelle T_D_Planung, in welcher wir eine Möbelkollektion mit den zu erwartenden Abverkäufen planen möchten. Dabei sollen sowohl bestehende Stammdaten, als auch Freitexteingaben zur Auswahl stehen, weshalb der SQL-Durchgriff herangezogen wird.

```
CREATE TABLE [dbo].[T_D_Planung](
	[RowID] [uniqueidentifier] NOT NULL,
	[Preissegment] [varchar](50) NULL,
	[Händler] [varchar](50) NULL,
	[Produkt] [varchar](50) NULL,
	[Kommentar] [nvarchar](250) NULL,
	[Wert] [float] NULL,
	[LastChangedBy] [varchar](50) NULL,
	[LastChanged] [datetime] NULL,
	[PeriodeID] [smalldatetime] NULL,

)
```

Zur Anzeige in DeltaMaster empfiehlt sich eine View, welche nur relevante Informationen anzeigt. Spoiler-Alarm: Wir werden im späteren Verlauf ohnehin eine View brauchen:

```
CREATE VIEW [dbo].[V_D_Planung] AS
/*
	Sicht für die Anzeige der Plantabelle in DeltaMaster
*/

SELECT
	RowID
	,Jahr
	,Preissegment
	,Händler
	,Produkt
	,Kommentar
	,Wert
FROM

	T_D_Planung
```

Die für die Tabellenpflege notwendigen Eingabeprozeduren lassen sich mittels folgenden Aufrufes generieren, sofern ein DeltaMaster ETL-Metamodell vorhanden ist:

```
EXEC P_BC_Generate_DeltaMasterTableProc @Tabname = 'V_D_Planung', @BaseTabName =
'T_D_Planung'
```

Anschließend kann auf dem entstandenen Datenmodell eine relationale DeltaMaster-Anwendung aufgebaut werden. Das notwendige Vorgehen wird in der DeltaMaster-Hilfe detailliert beschrieben. Nach wenigen Handgriffen ist so ein eingabefähiger SQL-Durchgriff gebaut:

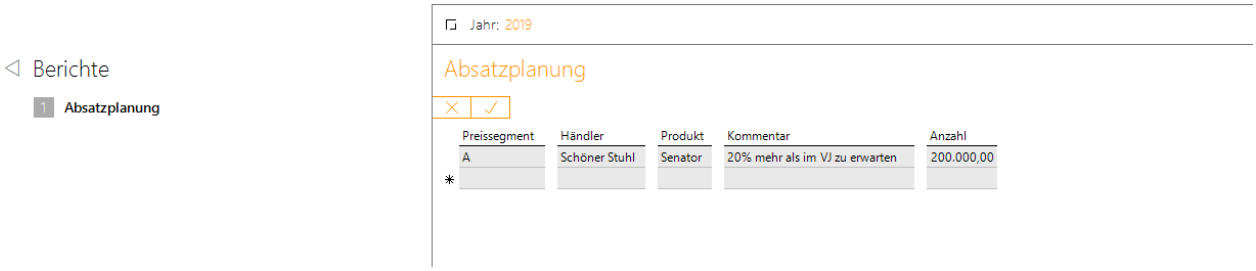


Abbildung 1: Eingabebericht

3 Vorjahreswerte durch Defaultauswahlen

Bei neu erfassten Eingaben kann es vorkommen, dass für die erfasste Kombination im Vorjahr bereits Werte vorhanden sind, die für die Eingabe von Relevanz sind. Diese Vorjahreswerte können in Delta-Master-Pivottabellen mit wenigen Handgriffen zur Anzeige gebracht werden. Beim SQL-Durchgriff ist etwas Erfindergeist gefragt.

Zuerst werden die Vorjahressummen in einer View so aggregiert, dass sie auf der Granularität des Eingabeberichtes vorliegen. Außerdem werden die Daten um ein Jahr verschoben, so dass z.B. die Ist-Daten für 2018 als Vorjahreswerte auf 2019 vorliegen:

```
CREATE VIEW V_D_Sum_Ist AS
/*
	Sicht für die Vorjahressummen des Absatzes.
	Werte werden um ein Jahr verschoben, um Join in
	V_D_Planung zu beschleunigen
*/

SELECT
	Jahr + 1 AS Jahr
	,Preissegment
	,Händler
	,Produkt
	,SUM(Wert) AS Wert
FROM
	T_Ist
GROUP BY
	Jahr
	,Preissegment
	,Händler
	,Produkt
```

Danach wird die eben erstellte View um eine Spalte zur Anzeige der Vorjahreswerte erweitert. Gleichzeitig wird dort auch die Ermittlung des Wertes eingebaut:

```
ALTER VIEW [dbo].[V_D_Planung] AS
/*
	Sicht für die Anzeige der Plantabelle in DeltaMaster
*/

SELECT
	RowID
	,p.Jahr
	,p.Preissegment
	,p.Händler
	,p.Produkt
	,p.Kommentar
	,p.Wert
	,s.Wert AS sumVJ
FROM
	T_D_Planung p

--Ermittlung Vorjahreswerte
LEFT JOIN V_D_Sum_Ist s
	ON p.Jahr = s.Jahr
	AND p.Preissegment = s.Preissegment
	AND p.Händler = s.Händler
	AND p.Produkt = s.Produkt
```

Für schon getätigte Eingaben und nach jeder Eingabe wird nun der Vorjahreswert in der entsprechenden Spalte angezeigt. Der Einwand „Ziel verfehlt" wäre hier aber durchaus angebracht. Damit zum Zeitpunkt der Eingabe innerhalb der Transaktion schon ein Vorjahreswert ermittelt wird lautet das Instrument der Wahl: Default-Wert. Mit der modifizierten Vorjahreswertermittlung aus der View können wir auch hier den benötigten Wert ermitteln (Einstellungen => Felder):

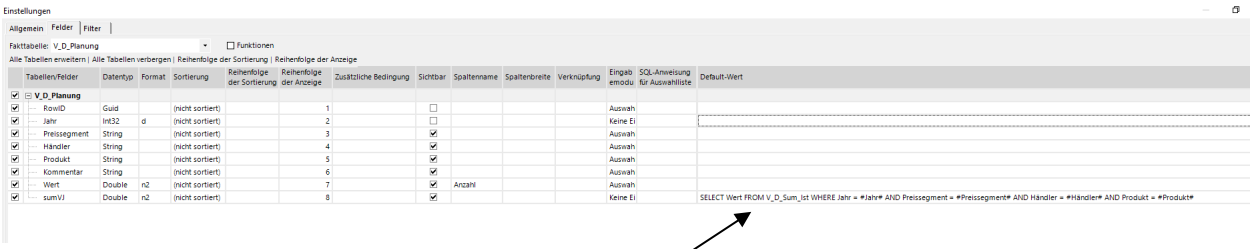


Abbildung 2: Konfiguration der Spalte

```
SELECT Wert FROM V_D_Sum_Ist WHERE Jahr = #Jahr# AND Preissegment = #Preissegment# AND
Händler = #Händler# AND Produkt = #Produkt#
```

Mittels der Platzhalter #Spaltenname# kann auf andere Spalten im SQL-Durchgriff referenziert werden. Charmant ist dabei, dass die Abfrage nach jeder Neueingabe auf einem Feld aktualisiert wird.

Abbildung 3: Eingabebericht mit Vorjahreswerten

Die Spalte „VJ" zeigt nun, wenn vorhanden, den Vorjahreswert an. Für diese Spalte ist unbedingt die Eingabe zu deaktivieren. Außerdem sollte sie nicht als Parameter an die Prozedur übergeben werden (Checkbox „Parameter").

4 Suchfelder für Dropdown-Menüs

Aktuell ist es in SQL-Durchgriffen nicht möglich, Dropdown-Menüs mit einer Wildcard-Suche zu durchsuchen. Gerade bei langen Listen kann dies aber notwendig sein. Abhilfe schafft hier ein Suchfeld, welches in der View ergänzt wird.

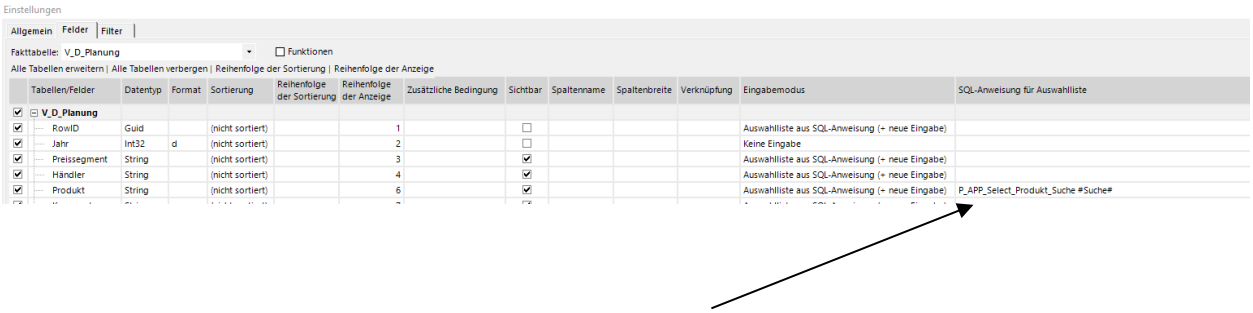
An dieser Stelle ein kleiner Exkurs in die Grundlagen von SQL-Eingabeberichten: Es ist möglich für die Dropdown-Auswahl spezielle SQL-Abfragen zu definieren, welche den Inhalt der Dropdown-Boxen bestimmen. Klassischerweise wird hier ein Select auf Dimensionselemente ausgeführt. Analog zu der beschriebenen Default-Auswahl kann auch an dieser Stelle mit Parametern gearbeitet werden, um die Dropdown-Auswahl abhängig vom Inhalt des Suchfeldes zu filtern. Dazu muss zunächst eine entsprechende Prozedur angelegt werden:

```
CREATE PROCEDURE P_APP_Select_Produkt_Search
(
	@SearchString VARCHAR(50) = ''
)
 AS
/*
	Prozedur zur suchfeldabhängigen Anzeige von Produkten im SQL-Durchgriff
	Wird der Parameter @SearchString nicht angegeben, werden alle Produkte angezeigt
*/

SELECT DISTINCT
	Produkt AS [ID]
FROM
	T_Ist i
WHERE

	Produkt LIKE '%' + @SearchString + '%'
```

Anschließend muss der Aufruf noch im SQL-Durchgriff eingerichtet werden (Einstellungen => Felder):




```
P_APP_Select_Produkt_Suche #Suche#
```

Gibt man nun einen Text im Feld „Produktsuche“ ein, wird die Dropdown-Liste der Spalte „Produkt“ entsprechend gefiltert. Die Spalte für das Suchfeld darf nicht als Parameter bei der Eingabe übergeben werden. Somit ist das Feld nach der Eingabe wieder leer und wird sinnvollerweise auch nicht an die Datenbank übertragen.

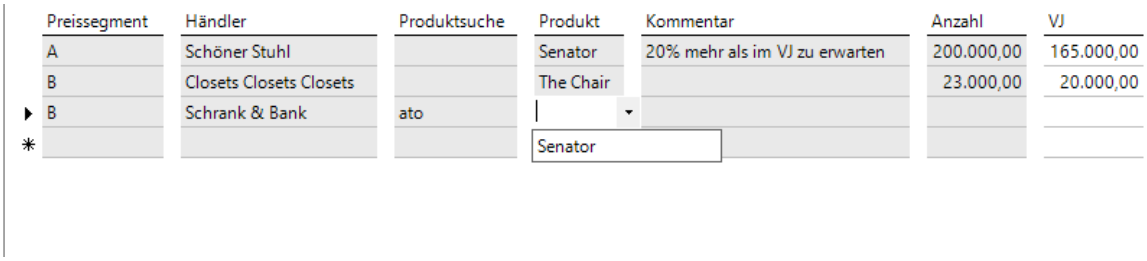


Abbildung 4: Suchfunktion

5 Summenzeilen

Ein weiterer Nachteil von SQL-Durchgriffen gegenüber Pivot-Tabellen ist die fehlende Aggregation der dargestellten Werte. Hier eine Summenzeile zu erzeugen ist deutlich aufwändiger, jedoch nicht unmöglich. Die komplette Dynamik eines All-Members in einem multidimensionalen Modell zu erreichen ist jedoch nicht das Ziel. Vielmehr soll eine Summenzeile auf der Granularität angezeigt werden, auf der Eingaben üblicherweise vorgenommen werden. Für das bisher verwendete Beispiel soll eine Summe für alle getätigten Eingaben angezeigt werden. Dazu wird wie bei den Vorjahreswerten eine View für die Summe erstellt:

```
CREATE VIEW V_D_Sum_Plan AS
/*
       Sicht für die Darstellung der Planwertsumme
*/

SELECT
       Jahr AS Jahr
       ,SUM(Wert) Wert
       , 1 AS SortPos
FROM
       T_D_Planung
GROUP BY
       Jahr
```

Anschließend muss die Eingabeview entsprechend um eine einfache Berechnung erweitert werden:

```
[…]
UNION ALL

SELECT
	NULL AS RowID
	,Jahr
	,NULL AS Preissegment
	,NULL AS Händler
	,NULL AS Produkt
	,'SUMME' AS Kommentar
	,Wert
	,NULL AS sumVJ
	,'' AS Suche
	,SortPos
FROM
	V_D_Sum_Plan
```

Damit die Summe im Bericht stets unten steht, wird eine neue Spalte „SortPos“ eingeführt, welche bei der Summenzeile 1, bei den anderen Zeilen immer 0 ist. Anschließend wird im SQL-Durchgriff nach diesem Kriterium sortiert:

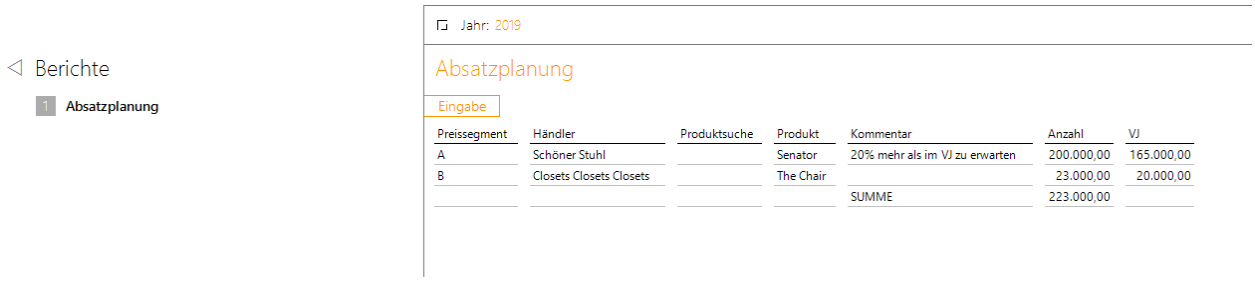


Abbildung 5: Summenzeile

Aktuell gibt es noch keine Möglichkeit, die Eingabe auf der Summenzeile zu deaktivieren. Dies muss entsprechend in der Eingabeprozedur abgefangen werden. Außerdem ist es an dieser Stelle nicht möglich, durch Defaultwerte eine Berechnung innerhalb der Transaktion durchzuführen. Die Summe wird somit erst nach Beenden der Eingabe aktualisiert.

6 Fazit

Dieser Blog gibt einen kurzen Überblick darüber, wie der SQL_Durchgriff mit ein paar Handgriffen um nützliche Funktionen erweitert werden kann. Defaultwerte und Auswahlprozeduren können dank der #Spaltenname#-Platzhalter trotz der sehr starren Tabellenstruktur eines SQL-Durchgriffs für einige Dynamik sorgen. Zusätzlich bietet die zugrundeliegende View eine zusätzliche Stelle, an der Logik untergebracht werden kann. Gepaart mit ein wenig Erfindergeist können so hilfreiche Anwendungen zur Datenpflege entstehen.