Einsatz analytischer Funktionen

Beim Aufbereiten von Faktentabellen für ein Data Warehouse sind vom ETL-Prozess immer wieder folgende Aufgaben zu lösen: Der Wert eines Kopfdatensatzes soll auf die zugehörigen Positionsdatensätze verteilt werden. Was sich zunächst einfach anhört, bereitet bei näherer Betrachtung oft Kopfzerbrechen, insbesondere wenn die Summe der verteilten Werte durch Rundungsdifferenzen nicht verfälscht werden soll. Dabei stellt sich die Frage, wie Rundungsdifferenzen ausgeglichen werden sollen. Zu beachten ist auch, dass für eine gleichmäßige Verteilung für jeden Kopfdatensatz zunächst die Anzahl der Positionen oder bei einer gewichteten Verteilung die Summe der Gewichte ermittelt werden muss.
Ein ETL-Prozess für die Lösung einer solchen Aufgabe kann sehr komplex werden. Kommen Lösungen mit Funktionen auf Basis prozeduraler Sprachelemente (wie z. B. Oracle PL/SQL) zum Einsatz, erweisen sich diese bei großen Datenmengen oft nicht als ausreichend performant. Versucht man hingegen ausschließlich mit SQL das Problem zu lösen, wird der ETL-Prozess aufgrund der Komplexität gerne in mehrere Einzelschritte zerlegt, die jeweils Zwischenergebnisse erzeugen. Das geht wiederum zu Lasten der Überschaubarkeit und damit der auch der Wartbarkeit.

Verschiedene Lösungsansätze

Deshalb suchen wir in dieser dreiteiligen Beitragsserie schrittweise nach einer Lösung, die im Kern auf SQL beruht und gleichzeitig einfach einzusetzen ist. Im ersten Teil wird zunächst eine reine SQL-Lösung vorgestellt, die vor allem auf analytischen SQL-Funktionen beruht, aber noch eine hohe Komplexität besitzt. Im zweiten Teil werden wir am Beispiel des SQL-Preprocessors der In-Memory-Datenbank EXASolution eine sehr elegante Umsetzung vorstellen, die sehr einfach angewendet werden kann. Im dritten Teil schließlich zeigen wir eine ebenso einfach einzusetzende Lösung mit der Oracle-Datenbank auf Basis von User-Defined Analytic Functions.

Ein Beispielszenario

Für die weiteren Betrachtungen wird das folgende Beispiel verwendet: Ein Online-Händler möchte die Versandkosten der Bestellungen auf die zugehörigen Bestellpositionen verteilen, so dass später die Marge pro Bestellposition berechnet werden kann. Der Einfachheit halber wird hier auf eine gewichtete Verteilung verzichtet. Es soll mit zwei Nachkommastellen gerechnet werden und Rundungsdifferenzen sollen möglichst kleinteilig auf die Positionen verteilt werden. Entsteht also eine Rundungsdifferenz von z. B. 0,02 EUR, so soll diese nicht allein einer Position zugeteilt werden, sondern auf zwei Positionen verteilt werden.
Die Bestellungen und deren Versandkosten (VK) sind in der Tabelle BESTELLUNG, die Bestellpositionen sind in der Tabelle BESTELLPOSITION gespeichert. Die folgende Abbildung zeigt die Ausgangslage und das gewünschte Verteilungsergebnis. *(siehe Tabellen 1,2 und 3)*

Erläuterungen

Die Skripte für die Erzeugung dieser Tabellen inklusive der Testdaten befinden sich am Ende dieses Betrags, so dass die hier dargestellten Beispiele nachvollzogen werden können.

Das SQL-Statement, das das gewünschte Ergebnis erzeugt, ist auf den ersten Blick schwer verständlich, deshalb zunächst ein paar Erklärungen zu einigen Teilkomponenten. *(siehe Listing 1)*

Gibt für eine Bestellposition an, um die wievielte Position einer Bestellung es sich handelt. *(siehe Listing 2)*

Gibt die Anzahl der Bestellpositionen pro Bestellung an. *(siehe Listing 3)*

Wird verwendet, um auf zwei Nachkommastellen zu runden. *(siehe Listing 4)*

Berechnet den gerundeten gleichmäßig verteilten Wert für die Versandkosten. *(siehe Listing 5)*

Steht für den kleinsten möglichen positiven Wert bei zwei Nachkomastellen, also 0,01. Dieser Wert wird verwendet um Rundungsdifferenzen möglichst kleinteilig zu verteilen.

SQL-Statement

Das gesamte SQL-Statement für das gewünschte Ergebnis sieht wie folgt aus. *(siehe Listing 6)*

Durch den umfangreichen Einsatz analytischer Funktionen kann also die Aufgabe gelöst werden. Der Code ist jedoch sehr komplex und damit nur schwer nachzuvollziehen. Durch zusätzliche Anforderungen wie z. B. eine gewichtete Verteilung würde der Code sogar noch umfangreicher und komplexer werden.

Wir können den obigen Code jedoch bei der EXASolution-Datenbank als Vorlage für die Definition einer eigenen analytischen Funktion nutzen. Wie das geht zeigen wir im zweiten Teil dieser Beitragsserie. Im dritten Teil werden wir dann eine entsprechende Lösung mit der Oracle-Datenbank vorstellen.

Der Code

Code für die Erzeugung der Beispieltabellen. *(siehe Listing 7)*

Tabelle 1:

Tabelle Bestellungen

Bearbeiten

BNR	VK
1	3.02
2	3.66
3	7.81

Tabelle 2:

Tabelle Bestellpositionen

Bearbeiten

BNR	POS	UMSATZ
1	1	10.5
1	2	9.85
1	3	5.5
1	4	8
2	1	3.79
2	2	2.99
2	3	2.99
3	1	8.75

Tabelle 3:

Gewünschtes Ergebnis:

Bearbeiten

BNR	POS	UMSATZ	VK-VERTEILT
1	1	10.5	3.02
1	2	9.85	3.02
1	3	5.5	3.03
1	4	8	3.03
2	1	3.79	3.66
2	2	2.99	3.67
2	3	2.99	3.67
3	1	8.75	7.81

Listing 1:

```
row_number () over (partition by p.bnr order by 1)
```

Listing 2:

```
count (p.pos) over (partition by p.bnr)
```

Listing 3:

```
cast (… as decimal (10, 2))
```

Listing 4:

```
cast (b.vk / count (p.pos) over (partition by p.bnr) as decimal (10, 2)
```

Listing 5:

```
1 / power (10, 2)
```

Listing 6:

```
select p.*
     , case
       when row_number () over (partition by p.bnr order by 1)
         <= trunc (abs (b.vk - (count (p.pos) over (partition by p.bnr)
          * cast (b.vk / count (p.pos) over (partition by p.bnr)
         as decimal (10,2)))) / (1 / power (10, 2)))
       then cast (b.vk / count (p.pos) over (partition by p.bnr)
         as decimal (10,2))
          + sign (b.vk - count (p.pos) over (partition by p.bnr)
          * cast (b.vk / count (p.pos) over (partition by p.bnr)
         as decimal (10,2))) * 1 / power (10, 2)
       when row_number () over (partition by p.bnr order by 1)
          = 1 + trunc (abs (b.vk - (count (p.pos) over (partition by p.bnr)
          * cast (b.vk / count (p.pos) over (partition by p.bnr)
         as decimal (10,2)))) / (1 / power (10, 2)))
       then cast (b.vk / count (p.pos) over (partition by p.bnr)
         as decimal (10,2))
          + mod (b.vk - count (p.pos) over (partition by p.bnr)
          * cast (b.vk / count (p.pos) over (partition by p.bnr)
         as decimal (10,2)), 1 / power (10, 2))
       else cast (b.vk / count (p.pos) over (partition by p.bnr)
         as decimal (10,2))
       end vk_verteilt
  from bestellung b
     , bestellposition p
 where b.bnr = p.bnr;
```

Listing 7:

```
create table bestellung
 (bnr number (9)
, vk  number (10, 2));

create table bestellposition
 (bnr    number (9)
, pos    number (4)
, umsatz number (10, 2));

insert into bestellung (bnr, vk) values (1, 12.1);
insert into bestellung (bnr, vk) values (2, 11);
insert into bestellung (bnr, vk) values (3, 7.81);

insert into bestellposition (bnr, pos, umsatz) values (1, 1, 10.5);
insert into bestellposition (bnr, pos, umsatz) values (1, 2, 9.85);
insert into bestellposition (bnr, pos, umsatz) values (1, 3, 5.50);
insert into bestellposition (bnr, pos, umsatz) values (1, 4, 8.00);
insert into bestellposition (bnr, pos, umsatz) values (2, 1, 3.79);
insert into bestellposition (bnr, pos, umsatz) values (2, 2, 2.99);
insert into bestellposition (bnr, pos, umsatz) values (2, 3, 2.99);
insert into bestellposition (bnr, pos, umsatz) values (3, 1, 8.75);

commit;
```