Nested Tables Types als Ergänzung zu Pivot XML

Thomas Strub
Logica Deutschland GmbH & Co. KG
Frankfurt

Schlüsselworte
Nested Tables, pivot, pivot xml, unpivot, collect, PL/SQL

Einleitung
Die Verknüpfung von vorverarbeiteten Daten stellt an manchen Stellen neue Anforderungen an die Entwickler von Berichten, dies vor allem bei handgeschriebenem SQL-Code.

Der Vortrag zeigt wie man mit Nested Table Types die Weiterverarbeitung von strukturierten Daten vereinfachen kann. Dabei wird auf die Businessanforderungen und die Technik eingegangen.

Der Vortrag beginnt mit einem kurzen Abriss der Pivotfunktion und zeigt was damit möglich ist. Danach werden die vergleichbaren Funktionen bei Nested Tables, collect für pivot und table(NTT) für unpivot gegenübergestellt.

Weiter werden einige Funktionen zur Weiterverarbeitung von Nested Table Types kurz vorgestellt davon wird bei einigen auf das Laufzeitverhalten eingegangen.

Exkurs pivot und pivot xml
Zur Vorstellung der Funktionen werden die Daten aus dem Oracle Schema OE verwendet.

```
Create view v_pivot_standard as
SELECT *
FROM
  (SELECT TO_CHAR(order_date,'YYYYMM') mon,
               order_mode,
               COUNT(order_id) anzahl,
               SUM(order_total) value
               FROM oe.orders
               GROUP BY TO_CHAR(order_date,'YYYYMM'),
               order_mode
  ) pivot (SUM (anzahl) anzahl
         , SUM(value) wert
          FOR order_mode IN ('direct' as "DIREKT", 'online' as "ONLINE"))
   ORDER BY mon
```

Ergebnis:

	MON	DIREKT_ANZAHL	DIREKT_WERT	ONLINE_ANZAHL	ONLINE_WERT
1	199007	3	61655.7	(null)	(null)
2	199603	1	5546.6	(null)	(null)
3	199703	1	310	(null)	(null)
4	199801	1	59872.4	2	100056.6

Für wenige und davor bekannte Spalten ist die Pivotfunktion gut geeignet.

Aber wenn man die Tabelle gerne transponiert hätte:

	199007	199603	199703
ANZAHL	3	1	1
WERT	61655,7	5546,6	310

wird das ganze etwas unhandlicher. Insbesondere, wenn die Monate dynamisch sind.

Für den Fall der dynamischen Liste von Elementen gibt es bei pivot xml die Möglichkeit mit „any“ alle Optionen zu wählen:

```
SELECT * FROM oe.orders;
SELECT *
FROM
  (SELECT TO_CHAR(order_date,'YYYYMM') mon,
    order_mode,
    COUNT(order_id) anzahl,
    SUM(order_total) value
  FROM oe.orders
  GROUP BY TO_CHAR(order_date,'YYYYMM'),
    order_mode
  ) pivot xml (SUM (anzahl) anzahl, SUM(value) wert FOR order_mode IN (any)
)
ORDER BY mon
```

Ergebnis:

	MON	ORDER_MODE_XML
1	199007	<PivotSet><item><column name = "ORDER_MODE">direct</column><column name = "ANZAHL">3...
2	199603	<PivotSet><item><column name = "ORDER_MODE">direct</column><column name = "ANZAHL">1...
3	199703	<PivotSet><item><column name = "ORDER_MODE">direct</column><column name = "ANZAHL">1...
4	199801	<PivotSet><item><column name = "ORDER_MODE">direct</column><column name = "ANZAHL">1...

Für den Fall mit 2 Optionen bringt das nicht gerade viel Mehrwert, aber wenn man hier die Selektion auf Monate macht sieht das schon anders aus:

```
SELECT *
FROM
  (SELECT TO_CHAR(order_date,'YYYYMM') mon,
    order_mode,
    COUNT(order_id) anzahl,
    SUM(order_total) value
  FROM oe.orders
  GROUP BY TO_CHAR(order_date,'YYYYMM'),
    order_mode
  ) pivot xml (SUM (anzahl) anzahl, SUM(value) wert FOR mon IN (any) )
ORDER BY order_mode
```

	ORDER_MODE	MON_XML
1	direct	<PivotSet><item><column name = "MON">199007</column><column name = "ANZAHL"...
2	online	<PivotSet><item><column name = "MON">199801</column><column name = "ANZAHL"...

Die Daten sind pivotisiert und mit den XML-Befehlen weiterverarbeitbar.

Datenauszug:

```
<PivotSet><item><column name = "MON">199007</column><column name =
"ANZAHL">3</column><column name =
"WERT">61655.7</column></item><item><column name =
"MON">199603</column><column name = "ANZAHL">1</column><column name =
"WERT">5546.6</column></item><item><column name =
"MON">199703</column><column name = "ANZAHL">1</column><column name =
"WERT">310</column></item>……</PivotSet>
```

Anstatt any kann man auch ein subselect verwenden, mit dem die Werteliste eingeschränkt wird.

Die Verarbeitung von xml-Daten ist mit Datenbank-Befehlen möglich, aber eine gezielte Veränderung von Werten ist etwas aufwändig. Da die Daten zwischen Datenbankstruktur und XML hin und her konvertiert werden müssen.

Pivot mit nested table und collect

Als zusätzliche Möglichkeit des pivotierens kann man auch collect verwenden.

Die im Beispiel verwendeten Typen:

```
create type pv_obj as object
 (monat varchar2(6),
  anzahl number,
  wert number);
 /
create type pv_ntt as table of pv_obj;
/
```

Das select ist vergleichbar mit der bekannten Methode für pivot aus 10g mit group by und sum(case when ….end) as alias

```
Create view v_pv_ntt as
SELECT order_mode ,
  CAST(collect(pv_obj(mon,anzahl,value))AS pv_ntt) AS my_ntt
FROM
  (SELECT TO_CHAR(order_date,'YYYYMM') mon,
    order_mode,
    COUNT(order_id) anzahl,
    SUM(order_total) value
  FROM oe.orders
  GROUP BY TO_CHAR(order_date,'YYYYMM'),
    order_mode
  )
GROUP BY order_mode;
```

Das Ergebnis ist vergleichbar mit dem Ergebnis des pivot xml:

	ORDER_MODE	MY_NTT
1	direct	HR.PV_NTT('HR.PV_OBJ('199910',5,70627.5)','HR.PV_OBJ('199802',3,151501.6)','
2	online	

Wobei es auch einige Nachteile gibt. Beispielsweise wird beim Anlegen von Tabellen mit nested tables eine store as Klausel verlangt. Deswegen funktioniert ein create table as select nicht. Ein `nested table MY_NTT store as T_PV_NTT_MY_NTT_NESTED_TAB return as value` wird notwendig.
Weiterhin sind Nested Tables bei globalen temp Tabellen nicht erlaubt. Deren Einsatz sollte man aber ohnehin auf die notwendigen Fälle beschränken.

Exkurs unpivot
Zur Weiterverarbeitung der Daten kann es manchmal hilfreich sein, die Daten wieder zu unpivotieren:

```
SELECT *
FROM
  (SELECT mon, direkt_wert, online_wert FROM v_pivot_standard
  ) unpivot include nulls
(wert FOR order_mode IN ( direkt_wert as 'direkt' ,online_wert as 'online')
);
```

	MON	ORDER_MODE	WERT
1	199007	direkt	61655.7
2	199007	online	(null)
3	199603	direkt	5546.6
4	199603	online	(null)
5	199703	direkt	310
6	199703	online	(null)

Hier gibt es bei den klassischen pivot Funktionen zwei Nachteile, zum einen gibt es keine Möglichkeit in einem Schritt wieder zwei Spalten zurück zu bekommen, zum anderen ist bei xml-Typen die SQL-Syntax etwas umfangreicher.

Unpivot mit Nested Tables:
Bei Nested Tables gibt es durch den `table()` Operator eine relativ elegante Methode für das unpivot:

```
select order_mode, m.* from v_pv_ntt, table(my_ntt) m
```

Ergebnis:

	ORDER_MODE	MONAT	ANZAHL	WERT
1	direct	199910	5	70627.5
2	direct	199802	3	151501.6
3	direct	199909	7	367489.1
4	direct	199904	1	1636
5	direct	199007	3	61655.7
6	direct	200008	1	2075.2

Hier sieht man einen Vorteil von Nested Tables, beide Felder Anzahl und Wert sind in einem Schritt hergestellt.

Falls man Werte hat, die nicht mit der Anzahl der Pivotelemente multipliziert werden sollen gibt es beim klassischen unpivot wie auch beim unpivot der Nested Tables die Möglichkeit über analytische Funktionen den entsprechenden Wert nur am ersten Datensatz zu hinterlegen:

```
case when rank() over (partition by monat order by rownum) = 1 then
'Suchbegriff' else null end
```

Das waren die notwendigen Grundlagen.

Weiterverarbeitung von Nested Tables:
Damit ergibt die Nutzung von Nested Tables die Möglichkeit etwas Bekanntes auf eine andere Art zu machen. Beim Austausch von Daten mit einer GUI über JDBC kann man den Nested Tables einfach eine variable Anzahl an Spalten erzielen. Dies wäre auch über das Laden von 2 Tabellen möglich, die in der Applikation gejoined werden, dafür sollten die Listen aber vorsortiert werden. Aber von Vorteil ist es meistens die joins in der Datenbank durchzuführen.

Ein Datenaustausch über XML ist wegen der teilweise doppelt notwendigen Konvertierung sicher nicht in jedem Fall vorzuziehen.

In der Datenverarbeitung in der Datenbank ergibt die Verwendung von Nested Tables weiterhin die Möglichkeit mit Operatoren und Funktionen Manipulationen an den Daten vorzunehmen.

Folgende Möglichkeiten mit jeweiligen Einsatzzwecken gibt es:

function(V1 in NTT) return number
    Summenbildung, Durchschnitt, Top-Wert, usw.
function(V1 in NTT, V2 number) return NTT
    Umsortierung
    Top-N.
function(V1 in NTT, V2 in NTT) return number
    Gewichtete Summe
function(V1 in NTT, V2 in NTT) return NTT
    Listenabgleiche
    Projektionen

Grenzen in der Anwendung durch Verknüpfung von mehreren Operatoren gibt es keine. Nur sollte man nicht vergessen, dass bei größeren Datenmengen der Kontextswitch zwischen SQL und PL/SQL-Engine die Laufzeit negativ beeinflussen kann.

Für die einfache Addition von identischen NTTs sollte man den multiset operator nutzen:

```
  Select            id
                  , NVL(a.MY_NTT, PV_NTT( PV_OBJ(null,null,null)))
   multiset union   NVL(b.MY_NTT, PV_NTT( PV_OBJ(null,null,null)))
   multiset except  PV_NTT( PV_OBJ(null,null,null)) union_ntt
from a full outer join b on (a.id = b.id)
```

Laufzeitvergleich gegenüber pivot:
Für den Laufzeitvergleich bietet sich eine über connect by erzeugte Tabelle an.

```
CREATE TABLE million_rows
NOLOGGING
AS
    SELECT MOD(TRUNC(DBMS_RANDOM.VALUE(1,10000)),4) AS pivoting_col
     ,      MOD(ROWNUM,4)+10                         AS grouping_col
     ,      DBMS_RANDOM.VALUE                        AS summing_col
     ,      RPAD('X',70,'X')                         AS padding_col
     FROM   dual
     CONNECT BY ROWNUM <= 500000;

Insert into million_rows
NOLOGGING
    SELECT MOD(TRUNC(DBMS_RANDOM.VALUE(1,10000)),10) AS pivoting_col
     ,      MOD(ROWNUM,10)+10                         AS grouping_col
     ,      DBMS_RANDOM.VALUE                        AS summing_col
     ,      RPAD('X',70,'X')                         AS padding_col
     FROM   dual
     CONNECT BY ROWNUM <= 500000;
```

Damit sind 1 Mio Zeilen Erzeugt, ein Pivot erzeugt daraus 10 Spalten und 10 Zeilen.

1 Mio Zeilen, 10 Gruppierungsspalten, 10 Zeilen, Pivot:

```
CREATE view v_pd_10z_pivot as
WITH pivot_data AS (
            SELECT pivoting_col
            ,      grouping_col
            ,      summing_col
            FROM   million_rows
            )
    SELECT *
    FROM   pivot_data
    PIVOT (SUM(summing_col) AS sum
   FOR    pivoting_col IN (0,1,2,3,4,5,6,7,8,9))
   ORDER  BY
          grouping_col;
```

10 rows selected

4,543ms elapsed
Plan hash value: 1201564532

```
-------------------------------------------------------------------------------
| Id  | Operation          | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |              |    10 |   240 |  5433   (1)| 00:01:06 |
|   1 |  SORT GROUP BY PIVOT|             |    10 |   240 |  5433   (1)| 00:01:06 |
|   2 |   TABLE ACCESS FULL | MILLION_ROWS |  1000K|    22M|  5404   (1)| 00:01:05 |
-------------------------------------------------------------------------------
```

1 Mio Zeilen, 10 Gruppierungsspalten, 10 Zeilen, collect:

```
CREATE view v_pd_10z_coll as
SELECT grouping_col ,
  CAST(collect(time_obj(pivoting_col,summing_col))AS time_ntt) AS my_ntt
FROM
  (SELECT pivoting_col,
    grouping_col,
    SUM (summing_col) summing_col
  FROM million_rows
  GROUP BY pivoting_col,
    grouping_col
  )
GROUP BY grouping_col;
```

10 rows selected

2,212ms elapsed
Plan hash value: 251560918

```
-------------------------------------------------------------------------------------
| Id  | Operation            | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |              |    10 |   280 |  5433   (1)| 00:01:06 |
|   1 |  SORT GROUP BY       |              |    10 |   280 |  5433   (1)| 00:01:06 |
|   2 |   VIEW               |              |    71 |  1988 |  5433   (1)| 00:01:06 |
|   3 |    HASH GROUP BY     |              |    71 |  1704 |  5433   (1)| 00:01:06 |
|   4 |     TABLE ACCESS FULL| MILLION_ROWS |  1000K|    22M|  5404   (1)| 00:01:05 |
-------------------------------------------------------------------------------------
```

Ergebnis:
Collect ist in dem Fall schneller als pivot bei mehreren Spalten und wenigen Zeilen.

Laufzeitvergleich (2):
Um das Verhalten bei größeren Datenmengen zu beobachten fügen wir in die bekannte Tabelle nochmal 500.000 Datensätze hinzu. Diesmal mit weiteren Einträgen für grouping_col

```
Insert into million_rows
NOLOGGING
    SELECT MOD(TRUNC(DBMS_RANDOM.VALUE(1,10000)),10) AS pivoting_col
     ,      MOD(ROWNUM,100000)+10                        AS grouping_col
     ,      DBMS_RANDOM.VALUE                          AS summing_col
     ,      RPAD('X',70,'X')                           AS padding_col
     FROM   dual
     CONNECT BY ROWNUM <= 500000;
```

1,5 Mio Zeilen, 100.000 Gruppierungsspalten, 10 Zeilen, Pivot:

```
Select * from v_pd_10z_pivot where rownum > 1

GROUPING_COL 0_SUM 1_SUM 2_SUM 3_SUM 4_SUM 5_SUM 6_SUM 7_SUM 8_SUM 9_SUM
------------ ----- ----- ----- ----- ----- ----- ----- ----- ----- -----
```

11,182ms elapsed
Plan hash value: 1180164315

```
-------------------------------------------------------------------------------------------------
| Id  | Operation              | Name         | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |              |   100K|    13M|       | 10254   (1)| 00:02:04 |
|   1 |  COUNT                 |              |       |       |       |            |          |
|*  2 |   FILTER               |              |       |       |       |            |          |
|   3 |    VIEW                |              |   100K|    13M|       | 10254   (1)| 00:02:04 |
|   4 |     SORT GROUP BY PIVOT|              |   100K|  2441K|    51M| 10254   (1)| 00:02:04 |
|   5 |      TABLE ACCESS FULL | MILLION_ROWS |  1500K|    35M|       |  5960   (1)| 00:01:12 |
-------------------------------------------------------------------------------------------------
```

1,5 Mio Zeilen, 100.000 Gruppierungsspalten, 10 Zeilen, Collect:

```
Select * from v_pd_10z_collect where rownum > 1

GROUPING_COL           MY_NTT
---------------------- ----------------
```

17,998ms elapsed
Plan hash value: 3730358821

```
-------------------------------------------------------------------------------------------------
| Id  | Operation              | Name         | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |              |   100K|  3417K|       | 13079   (1)| 00:02:37 |
|   1 |  COUNT                 |              |       |       |       |            |          |
|*  2 |   FILTER               |              |       |       |       |            |          |
|   3 |    VIEW                |              |   100K|  3417K|       | 13079   (1)| 00:02:37 |
|   4 |     SORT GROUP BY      |              |   100K|  2832K|       | 13079   (1)| 00:02:37 |
|   5 |      VIEW              |              |   707K|    19M|       | 13079   (1)| 00:02:37 |
|   6 |       HASH GROUP BY    |              |   707K|    16M|    51M| 13079   (1)| 00:02:37 |
|   7 |        TABLE ACCESS FULL| MILLION_ROWS |  1500K|    35M|       |  5960   (1)| 00:01:12 |
-------------------------------------------------------------------------------------------------
```

Predicate Information (identified by operation id):

```
---------------------------------------------------
   2 - filter(ROWNUM>1)
```

Hier sieht man, dass das klassische Pivot schneller ist. Dies bedeutet, dass man nie aus kleinen Datenmengen auf das endgültige Verhalten schließen kann.

Weiterverarbeitung der Daten.
Für die weitere Berechnung werden die Ergebnisse der beiden Views in Tabellen materialisiert.

`p_table` für den Inhalt aus `v_pd_10z_pivot`

```
create table p_table
(grouping_col number
, v0 number
, v1 number
, v2 number
, v3 number
, v4 number
, v5 number
, v6 number
, v7 number
, v8 number
, v9 number);

insert into p_table
       select * from v_pd_10z_pivot;
```

und

`t_ntt_pivot` für den Inhalt aus `v_pd_10z_collect`

```
create table t_ntt_pivot
(grouping_col number
, my_ntt time_ntt)
 nested table my_ntt store as m_t_ntt return value;

insert into t_ntt_pivot
       select * from v_pd_10z_collect;
```

100.000 Zeilen, Summe aus 10 Feldern pivot:

```
select * from (
  select grouping_col,
     nvl(v0,0)+nvl(v1,0)+nvl(v2,0)+nvl(v3,0)+nvl(v4,0)
    +nvl(v5,0)+nvl(v6,0)+nvl(v7,0)+nvl(v8,0)+nvl(v9,0)total
  from p_table
) where rownum >1;

GROUPING_COL            TOTAL
----------------------- -----------------------
```

266ms elapsed
Plan hash value: 150978838

```
-------------------------------------------------------------------------------
| Id  | Operation           | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |         | 90431 |    12M|   410   (1)| 00:00:05 |
|   1 |  COUNT              |         |       |       |            |          |
|*  2 |   FILTER            |         |       |       |            |          |
|   3 |    TABLE ACCESS FULL| P_TABLE | 90431 |    12M|   410   (1)| 00:00:05 |
-------------------------------------------------------------------------------
```

100.000 Zeilen, Summe aus 10 Feldern collect:

Für die einfache Aggregation kann man folgende Funktion nutzen:

```
create or replace function FCT_SUM_VALUES2(input_ntt time_ntt) return
number is
    v_p number := 0;
begin
select nvl(sum(wert ),0) into v_p
                    from table(input_ntt);
return v_p;
end;
```

Folgendes select ergibt das gleiche Ergebnis:

```
select * from (
   select t.grouping_col, FCT_SUM_VALUES2(my_ntt)total from t_ntt_pivot t)
where rownum > 1;
```

```
GROUPING_COL            TOTAL
----------------------- -----------------------
```

552ms elapsed
Plan hash value: 3321699824

```
-------------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name                    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                         | 98539 |  2213K|   137   (1)| 00:00:02 |
|   1 |  TABLE ACCESS BY INDEX ROWID| M_T_NTT                 |  4346 |   152K|     1   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN           | SYS_FK0000095194N00002$ |  1738 |       |     1   (0)| 00:00:01 |
|   3 |  COUNT                       |                         |       |       |            |          |
|*  4 |   FILTER                     |                         |       |       |            |          |
|   5 |    TABLE ACCESS FULL         | T_NTT_PIVOT             | 98539 |  2213K|   137   (1)| 00:00:02 |
-------------------------------------------------------------------------------------------------------------
```

Die Laufzeit mit den NTT ist in dem Fall wieder höher. Insbesondere der Lookup von jeder Zeile in der angehängten Tabelle kostet etwas Zeit. Hier gibt es wieder den Trade Off zwischen les- und wartbarem Code und Laufzeit.

Laufzeitvergleich – einzelne Funktion vs Aufruf von verschachtelten Funktionen:
Beim verknüpfen von mehreren Operatoren sollte man immer beachten, dass bei verschachtelten Aufrufen mehrere Kontextswitche notwendig sind, wenn man eine Funktion kompakt schreibt aber nicht.

Fazit
Mit Nested Tables gibt es zusätzlich zu der pivot Funktionalität eine weitere Methode spaltenorientiert Daten zu verarbeiten.

Insbesondere wenn sich die Anzahl der Spalten dynamisch ist oder viele Operationen auf den schon aggregierten Zahlen vorgenommen werden müssen bietet es sich an über Nested Tables nachzudenken.

Für die Datenverarbeitung rein in der Datenbank macht aber ein normalisiertes Datenmodell oftmals mehr Sinn.

Für die Nutzung im Frontend gibt es keine Einschränkungen. Über jdbc lassen sich ohne größere Probleme cursor mit Nested Tables Types auslesen.

Kontaktadresse:
Thomas Strub
Logica Deutschland GmbH & Co. KG | Now part of CGI
Am Limespark 2
D-65843 Sulzbach(Taunus)

Telefon: +49 151 40234650
E-Mail thomas.strub@logica.com | thomas.strub@cgi.com
Internet: www.logica.de | www.cgi.com