TOP TIP – by Tim Onions, TOdC Limited

Those of you with elephantine memories will recall that Top Tips used to be a regular feature in previous issues. I was very pleased, therefore, when Oracle Scene contacted me and asked if I had anything that could be used to re-start the feature. Previously the tips came from many sources but time constraints with this issue mean that this first one is sourced from me alone. I very much hope that in future sufficient interest and feedback will be generated for me to act more as a compiler of tips rather than the sole author of them. If you do have any tips (and they don't just have to be technical ones), please send them on to Oracle Scene at the email address you can find at the front of the magazine.

Analytic functions are simply awesome and I commend them to you whole heartedly! Various features have been covered before but recently I made good use of the `NTILE` function. The `NTILE` analytic function will evenly divide the resulting rows into N approximate sets (where N is the number of sets you wish to have the results divided up into) and assign a tile value to each row returned. So what this means is if I have a table of 20 rows and need to split the data into equal sets of 5 rows per set I use `NTILE(4)`. Furthermore, if I want to specify the ordering rules for which the data should be allocated to the tiles I can do so via an order by clause.

The official documentation describes the features of this function as being:

```
NTILE(expr) OVER ([query_partition_clause] order_by_clause)
```

Where:

`expr` is the number of sets we require
`query_partition_clause` allows you to tile within subsets of your data (the partitions)
`order_by_clause` defines the ordering rules to determine which tile data goes into

All this is going to be best explained with some examples.

Example 1 – split a table into 4 equal sub-sets

This example is based on one way I use `NTILE`. I needed to perform some processing in parallel (the usual story, overnight batch work overrunning the weekend). I already had a procedure that took start and end ids so it was easy to run 4 of these in parallel. However, I needed a quick and easy way to split the data to be processed into equal parts, never having to worry if more data got added in between runs. Using a very basic `NTILE` query the starting point was to get the rows back with a `NTILE` value between 1 and 4 for each row. This is done like so (for space reasons let's assume my table as only 8 rows, with one column ItemId taking values 1 through 8):

```
SELECT ItemID
      ,NTILE(4) OVER (ORDER BY ItemID) AS Tile
FROM  DataItemTable;
```

And representative results would look like:

```
ItemID      Tile
------      ----
1           1
2           1
3           2
4           2
5           3
6           3
7           4
8           4
```

Knowing what these results look like I can now use the initial query as an inline view and get the start/end ItemIDs for any Tile – so to get the data for Tile 3 the query is:

```
SELECT MIN(v.ItemId) AS mn
      ,MAX(v.ItemId) AS mnx
FROM  (SELECT ItemID
            ,NTILE(4) OVER (ORDER BY ItemID) AS Tile
      FROM  DataItemTable) v
WHERE v.Tile=3
GROUP BY v.Tile;
```

```
TILE  MN    MX
----  --    --
   3   7     8
```

(To get the 4 rows of data for all 4 tiles simply drop the `WHERE v.Tile=3 clause)`

Notice that here we do not have a partition clause, without this we allow the `NTILE` function to work over the entire range of resultset values returned.

Example 2 – get the upper and lower tiles per region based on sales

Here we use the partition clause to direct the `NTILE` to work within a subset of the data. Imagine a SalesData table with region, sales rep and sales data (this is you see is a totally made up example). We want to show the top set of and the bottom set of sales-people based on sales (ie split into 2 equal tiles, good and bad!) per region. Clearly we need to order by the amount of sales each rep has but we also want the tiling to be done within region not over the entire table (actually we can do both but the point of this example is to show the use of the partition clause).

```
SELECT Region
      ,SalesRepName
      ,Sales
      ,NTILE(2) OVER (PARTITION BY Region ORDER BY Sales DESC)
            AS RegionTile
      ,NTILE(2) OVER (ORDER BY Sales) AS CompanyTile
FROM  SalesData
ORDER BY Region,RegionTile;
```

Which results in the following output:

```
REGION     SALESREPNNAME  SALES REGIONTILE COMPANYTILE
---------- -------------- ----- ---------- -----------
East       Bloggs           250          1           2
East       Does              50          1           1
East       Jones             15          2           1
East       Smith             10          2           1
North      Does             500          1           2
North      Jones            150          1           2
North      Bloggs            50          2           1
North      Smith            100          2           1
South      Smith           1000          1           2
South      Jones           1050          1           2
South      Bloggs             5          2           1
South      Does             100          2           1
West       Bloggs           500          1           2
West       Smith           1000          1           2
West       Does              50          2           1
West       Jones            150          2           2
```

The sample data used for this example is:

```
INSERT INTO salesdata VALUES ('North','Smith',100);
INSERT INTO salesdata VALUES ('North','Jones',150);
INSERT INTO salesdata VALUES ('North','Bloggs',50);
INSERT INTO salesdata VALUES ('North','Does',500);
INSERT INTO salesdata VALUES ('South','Smith',1000);
INSERT INTO salesdata VALUES ('South','Jones',1050);
INSERT INTO salesdata VALUES ('South','Bloggs',5);
INSERT INTO salesdata VALUES ('South','Does',100);
INSERT INTO salesdata VALUES ('East','Smith',10);
INSERT INTO salesdata VALUES ('East','Jones',15);
INSERT INTO salesdata VALUES ('East','Bloggs',250);
INSERT INTO salesdata VALUES ('East','Does',50);
INSERT INTO salesdata VALUES ('West','Smith',1000);
INSERT INTO salesdata VALUES ('West','Jones',150);
INSERT INTO salesdata VALUES ('West','Bloggs',500);
INSERT INTO salesdata VALUES ('West','Does',50);
```

***Disclaimer**: You must always check the hints, tips and scripts presented in this paper before using them and always try them out on a test database before running against a live system. Whilst every care has been taken to ensure the scripts function properly and are totally unobtrusive and benign (when used properly), neither the authors nor TOdC Limited can take any responsibility or liability for what effect they have when you use them. All that being said TOdC Limited and the authors have successfully used all the material presented on this site for many years without encountering any serious issues.*

Tim Onions is an independent database consultant with over 15 years of experience with Oracle databases. He specializes in the architecture, application design and database design of high performance systems, as well as tuning and optimisation techniques together with technical management. Tim can be contacted via Tim.Onions@TOdC.co.uk. The scripts described in this article are available for download from the website http://www.TOdC.co.uk.