Table of Contents

1. Using Cube to generate a rows table

There are times when it is helpful to have a table which has a certain number of rows. It seems simple but there are times that this comes in handy.

You could just define the table and then do a lot of inserts- but that seems boring particularly if you need a lot of rows. This discusses one way to generate the rows for that table. It is not the most efficient but it does give us some more work with analytical techniques.

We will start with a simple demo to show you how you might use a number table. Assume we have a table MyNbrs which contains the numbers from 1 to 50. The code is in the demo. It hard codes the numbers.

Now we want a set of dates for the month of November 2012. We can start with the last day of October and add 1 then 2, then 3 etc to build dates. We want to stop with 30 dates

Demo 01: Using a numbers table to build a calendar. We could insert these values into a table or just use them in a result set.

```
select date '2012-10-31'+ col
from myNbrs
where col <= 30;
```

```
DATE'2012
---------
01-NOV-12
02-NOV-12
03-NOV-12
04-NOV-12
05-NOV-12
06-NOV-12
. . .  rows skipped
28-NOV-12
29-NOV-12
30-NOV-12

30 rows selected.
```

But setting up a table of 365 values or 10,000 values would be tedious if we coded each insert. We want to build that table. We don't have to worry too much about efficiency since we could just create the table once and then use it whenever we need a series of numbers.

2. Setting up a table with a set number of rows

Demo 02: Create the table and insert the first value- 1

```
create table starter ( col integer);
insert into starter values (1);
```

Demo 03: If we add a group by cube we get two rows- they both have the value 1, but we have two numbers

```
select 1 as Num1
from starter
group by cube(col);
```

```
      Num1
-----------
          1
          1
2 rows selected
```

Demo 04: Now we group by cube(col,col) and we get 4 rows.

```
select 1 as Num1
from starter
group by cube(col, col);
```

```
      Num1
-----------
          1
          1
          1
          1
4 rows selected
```

Demo 05: With a three way group we get 8 rows; with 4 we would get 16, with 5 we would get 32- binary magic!

```
select 1 as Num1
from starter
group by cube(col, col, col);
```

```
      Num1
-----------
          1
          1
          1
          1
          1
          1
          1
          1
8 rows selected
```

Demo 06: We always get the value 1; the more important part is how many rows we get. If we use the generating query as a table expression in the from clause, we can just get the number of rows.

```
select 1 as Num1
from starter
group by cube(col, col, col, col);

select count(*) as NumberRows
from (
   select 1 as Num1
   from starter
   group by cube(col, col, col, col)
)tbl;
```

```
NumberRows
-----------
         16
```

```
select 1 as Num1
from starter
group by cube(col, col, col, col, col);
select count(*)as NumberRows
from (
   select 1 as Num1
   from starter
   group by cube(col, col, col, col, col)
)tbl;
```

```
NumberRows
-----------
         32
```

I hope you see the pattern; with each new value in the cube function we double the number of rows.
With 8 arguments to cube, you get 256 rows; with 12 you get 4096 which you should recognize as the binary base numbers. This is one of those patterns that you can use without fully understanding it.

We can use that query to do inserts into a table.

Demo 07: With these statements you get 32 rows into the table MyRows. Each row has a column with a value of 1

```
create table MyRows ( col integer);
insert into MyRows
     select 1 as Num1
     from starter
     group by cube(col, col, col, col, col);
select * from myRows ;
```

Do we really need that table starter? It only serves to provide the value to use in the generated result set. What about a subquery?

Demo 08: A result set with 64 rows, all with the value 13

```
select 13 as num
from (
      select 1 AS num2 from dual
      ) tbl2
group by cube(num2, num2, num2, num2, num2, num2);
```

3. Generating numbers

Demo 09: This probably still looks like a trick; we need something useful. How about generating numbers from 101 to 116?

```
select num + ROW_NUMBER()over (order by num) as the_count
from   (
   select 100 as num
   from  (
      select 1 AS num2 from dual
      ) tbl2
   group by cube(num2, num2, num2, num2)
   ) tbl;
```

```
           the_count
--------------------
                 101
                 102
                 103
                 104
                 105
                 106
                 107
                 108
                 109
                 110
                 111
                 112
                 113
                 114
                 115
                 116

16 rows selected
```

The following query takes a while to run, but we can get a set of numbers in a certain range. Adjust the group by cube to generate enough numbers and then use a filter to set the upper range.

Demo 10: This query generates the numbers from 101 to 3000; this is 2900 rows and took 15 seconds on my system. The CTE generates 4096 rows based on the Group by cube. That is the part of the query that takes time.

```
with numbers as (
select num + ROW_NUMBER()over (order by num) as the_count
from
   (
   select 100 as num
   from
      (
      select 1 AS num2 from dual
      ) tbl2
   group by cube(num2, num2, num2, num2, num2, num2, num2, num2, num2, num2,
num2, num2)
   ) tbl
)
select the_count
from numbers
where the_count <= 3000;
```

This is not a great approach for generating a lot of numbers if you want to use this as part of a query. You could insert the values into a number table so that you take the hit once for generating the numbers. But if you need only a few rows, using this technique is sensible.

4. Building a calendar

We have 18 weeks in a semester. We need to generate at least 18 rows; 16 is not enough so we need 32 rows- that is 5 parameters to group by cube.

Demo 11:

```
with numbers as (
select num + ROW_NUMBER()over (order by num) as the_count
from
   (
   select 0 as num
   from
      (
      select 1 AS num2 from dual
      ) tbl2
   group by cube(num2, num2, num2, num2, num2)
   ) tbl
)
select the_count, 'Unit: ' || the_count as Units
from numbers
where the_count <= 18 ;
```

```
THE_COUNT            UNITS
-------------------- ------------------------------------------------
1                    Unit: 1
2                    Unit: 2
3                    Unit: 3
4                    Unit: 4
5                    Unit: 5
6                    Unit: 6
7                    Unit: 7
8                    Unit: 8
9                    Unit: 9
10                   Unit: 10
11                   Unit: 11
12                   Unit: 12
13                   Unit: 13
14                   Unit: 14
15                   Unit: 15
16                   Unit: 16
17                   Unit: 17
18                   Unit: 18

 18 rows selected
```

Demo 12: What were the weekend days in the spring 2016 semester? (You remember weekends!)

```
Variable startdtm  varchar2(10);
Exec :startdtm  := '2016-01-18';

Variable stopdtm  varchar2(10);
Exec :stopdtm  := '2016-05-24';

With weekends as(
select  dtm
from (
```

```
   select to_date(:startdtm, 'yyyy-mm-dd') + ROW_NUMBER()over (order by num)
as dtm
   from
     ( select 1 as num
       from (select 1 as n from dual) tbl0
       group by cube(n, n, n, n, n, n, n, n)
      ) tbl
   ) tbl2
where  dtm <= to_date(:stopdtm, 'yyyy-mm-dd') and to_char(dtm, 'fmday') in
('sunday', 'saturday')
)
select dtm
from weekends ;
```

```
DTM
---------
23-JAN-16
24-JAN-16
30-JAN-16
31-JAN-16
06-FEB-16
07-FEB-16
13-FEB-16
14-FEB-16
20-FEB-16
21-FEB-16
27-FEB-16
28-FEB-16
05-MAR-16
06-MAR-16
12-MAR-16
13-MAR-16
19-MAR-16
20-MAR-16
26-MAR-16
27-MAR-16
02-APR-16
03-APR-16
09-APR-16
10-APR-16
16-APR-16
17-APR-16
23-APR-16
24-APR-16
30-APR-16
01-MAY-16
07-MAY-16
08-MAY-16
14-MAY-16
15-MAY-16
21-MAY-16
22-MAY-16

36 rows selected.
```

Demo 13: What happens with this month? (http://en.wikipedia.org/wiki/1582) But read the output first!

```
Variable startdtm  varchar2(10);
Exec :startdtm  := '1582-10-01';

Variable stopdtm  varchar2(10);
Exec :stopdtm  := '1582-10-31';

With c as(
select  dtm
from (
   select to_date(:startdtm, 'yyyy-mm-dd') - 1 + ROW_NUMBER()over (order by
num)  as dtm
   from
     ( select 1 as num
       from (select 1 as n from dual) tbl0
       group by cube(n, n, n, n, n, n, n, n)
      ) tbl
   ) tbl2
where  dtm <= to_date(:stopdtm, 'yyyy-mm-dd')
)
select to_char(dtm, 'yyyy-mm-dd') as OCT1582
from c ;
```

5. Using a calendar

Suppose we want to get the number of orders for each day in June 2015. We are using the oe_order_headers table for this. If we look at just the order date in the oe_order headers table for June 2015, we get 12 rows. That's means we have a lot of days with no orders.

```
select order_id, order_date
from oe_orderheaders
where to_char(order_date, 'yyyy-mm') = '2015-06';
```

```
    ORD_ID ORD_DATE
---------- ---------
       540 02-JUN-15
       390 04-JUN-15
       395 04-JUN-15
       301 04-JUN-15
       302 04-JUN-15
       303 10-JUN-15
       306 04-JUN-15
       307 04-JUN-15
       312 07-JUN-15
       313 07-JUN-15
       324 11-JUN-15
       378 14-JUN-15

12 rows selected.
```

Demo 14: This gives us the counts only for days where we have orders.

```
select order_date , count(*) from oe_orderheaders
 where to_char(order_date, 'YYYY-mm') = '2015-06'
group by order_date;
```

```
ORD_DATE    COUNT(*)
--------- ----------
04-JUN-15          6
```

```
11-JUN-15          1
10-JUN-15          1
02-JUN-15          1
07-JUN-15          2
14-JUN-15          1

6 rows selected.
```

So how do we get a count- 0 for each day with no orders.

Demo 15: We build a June 2015 calendar and left join it to the order headers table. We can use a CTE to build the calendar and another CTE to get the counts for each day with orders and then join the two CTEs.

```
variable startdtm  varchar2(10);
exec :startdtm  := '2015-06-01';

variable stopdtm  varchar2(10);
exec :stopdtm  := '2015-06-30';

With calJune2015 as(
   select dtm
   from (
      select to_date(:startdtm, 'yyyy-mm-dd') - 1 + ROW_NUMBER()over (order by num)
             as dtm
      from(
        select 1 as num
        from (select 1 as n from dual) tbl0
        group by cube(n, n, n, n, n, n, n, n)
        ) tbl
      ) tbl2
where  dtm <= to_date(:stopdtm, 'yyyy-mm-dd')
)
,
ordersJune2015 as (
   select order_date , count(*)  OrderCount
   from oe_orderheaders
   where to_char(order_date, 'YYYY-mm') = '2015-06'
   group by order_date
)
select to_char(dtm, 'yyyy-mm-dd') as JuneDates, OrderCount
from calJune2015
left join  ordersJune2015 on calJune2015.dtm = ordersJune2015.order_date
order by JuneDates;
```

```
JUNEDATES  ORDERCOUNT
---------- ----------
2015-06-01
2015-06-02          1
2015-06-03
2015-06-04          6
2015-06-05
2015-06-06
2015-06-07          2
2015-06-08
2015-06-09
2015-06-10          1
2015-06-11          1
2015-06-12
2015-06-13
2015-06-14          1
2015-06-15
2015-06-16
```

```
2015-06-17
2015-06-18
2015-06-19
2015-06-20
2015-06-21
2015-06-22
2015-06-23
2015-06-24
2015-06-25
2015-06-26
2015-06-27
2015-06-28
2015-06-29
2015-06-30

30 rows selected.
```