Make Your Data Dance:
UNPIVOT, PIVOT and GROUP BY Extensions

Stew Ashton
Independent
Paris, France

Keywords:

SQL, PIVOT, UNPIVOT, GROUP BY

Summary

Reporting, Data Warehouses, Analytics, data migration: so much if it is data manipulation. When querying or copying, how often do we need to move data from columns to rows, from rows to columns, or aggregate at multiple levels?

This session will focus on the low code, very high performance SQL functions available in the Oracle Database:
- UNPIVOT: move data from multiple columns to multiple rows with one pass through the data, eliminating needless table scans.
- PIVOT: aggregate and move data from rows to columns.
- GROUP BY extensions (ROLLUP and CUBE among others): aggregate at multiple levels with one statement and one pass through the data.

Among the use cases presented will be:
- unpivoting start and end dates for analysis of temporal data;
- producing reports with totals and subtotals across rows and / or columns;
- matrix manipulation.

Preface

If you are reading this, you must think these pre-conference abstracts are worthwhile. I hope so, since they are extra work that is only used once (for DOAG), in contrast to the presentation itself that I expect to reuse at other venues.

This manuscript is intended to be “spoiler-free”: reading it is not a substitute for attending the presentation. Its purpose is to help you decide whether attending is a good idea for you - assuming that the summary above is not sufficient.

The intended audience of the session is “SQL practitioners”: people who write SQL as part of their work, or who tell SQL writers how to do their job.

Introduction

"Aggregate" functions return a single result row based on groups of rows, rather than on single rows. With the GROUP BY clause, one row is returned per group. Without the GROUP BY clause, one row is returned, even when there are no queried rows.

The Data Warehousing Guide explains Oracle extensions to GROUP BY that calculate multiple levels of aggregation in one pass. Queries that use these extensions (ROLLUP, CUBE or GROUPING SETS) are simpler and much more efficient than using UNION ALL.

For reporting and data manipulation, the same Guide presents the UNPIVOT and PIVOT operations, which convert columns to rows or rows to columns. The PIVOT also performs an implicit GROUP BY.

The presentation will present use cases for each function, both separately and in tandem.

UNPIVOT

The UNPIVOT operator rotates data - and metadata - from columns into rows. Look at this example:

```
drop table mytype purge;
create table mytype(
  pk number, code1 varchar2(5), code2 varchar2(5),
             code3 varchar2(5), code4 varchar2(5)
);
insert into mytype values(1, '111', '112', '113', NULL);
insert into mytype values(2, '112', '113', '111', NULL);

select * from mytype
unpivot(code for col in(code1, code2, code3, code4));

        PK COL   CODE
---------- ----- -----
         1 CODE1 111
         1 CODE2 112
         1 CODE3 113
         2 CODE1 112
         2 CODE2 113
         2 CODE3 111
```

The CODE1 through CODE4 columns now each belong to separate rows (by default, NULL values are ignored but we can say they should be included). Not only that, but the column names (which are metadata) have now become data.

We could get the same result from a series of SELECTs with UNION ALL, but the advantage of UNPIVOT is that only one scan of the data takes place.

The presentation will show a common use case: combining start and end points of date ranges in order to get all the date boundaries.

PIVOT

The PIVOT operator does an implicit GROUP BY and rotates data from multiple rows into one row with multiple columns. Whereas UNPIVOT converts column names to data, PIVOT transforms data into column names.

```
select * from (
  select deptno, job, sal from emp
)
pivot(avg(sal) for job in (
  'ANALYST',
  'CLERK',
  'MANAGER',
  'PRESIDENT',
  'SALESMAN' as SALESMAN
));
```

```
    DEPTNO  'ANALYST'    'CLERK'  'MANAGER' 'PRESIDENT' SALESMAN
---------- ---------- ---------- ---------- ----------- --------
        30                   950       2850                1400
        20       3000        950       2975
        10                  1300       2450        5000
```

- The data being pivoted must come from an aggregate function.
- The list of jobs must be known beforehand, since it determines what columns will appear in the final result set.
- The column list will generally need aliases (such as for SALESMAN) in order to generate normal column names.
- The presentation will later show pivoting of multiple data columns.

Transposing data with UNPIVOT and PIVOT

We sometimes want completely transpose columns to rows and rows to columns. The presentation will show UNPIVOT followed by PIVOT in the same SELECT, including an advanced solution that unpivots and pivots multiple columns (problem statement from Nimish Garg).

Matrix manipulation

A matrix is **displayed** in a tabular format, as in a spreadsheet: row, col1, col2,…, coln.

It is much easier to manipulate the data and store it correctly in the format: row_number, column_number, cell_value.

UNPIVOT can easily convert the tabular format to the preferred format, either for data migration or ease of processing. The result can easily be displayed in tabular form using PIVOT.

ROLLUP, CUBE, GROUPING SETS

GROUP BY extensions produce multiple levels of aggregation with one pass through the input data. Here is a use case with CUBE:

Suppose the requirement is to sum salaries by DEPTNO and JOB and to display them as follows:

JOB	10	20	30	TOTAL
ANALYST		6000		6000
CLERK	1300	1900	950	4150
MANAGER	2450	2975	2850	8275
PRESIDENT	5000			5000
SALESMAN			5600	5600
Total	8750	10875	9400	29025

- The white cells with numbers contain subtotals by DEPTNO and JOB;
- the yellow cells (right hand column) contain subtotals by JOB;
- the blue cells (bottom row) contain subtotals by DEPTNO;
- and the green cell (bottom right) contains the grand total.

The CUBE extension to GROUP BY is ideal for this kind of cross-tabular report: it will generate everything we need with one SELECT and one table scan.

```
select deptno, job, sum(sal) sal
from scott.emp
group by cube(deptno, job);
```

JOB	SAL
	29025
CLERK	4150
ANALYST	6000
MANAGER	8275
SALESMAN	5600
PRESIDENT	5000
	8750
CLERK	1300
MANAGER	2450
PRESIDENT	5000

	10875
CLERK	1900
ANALYST	6000
MANAGER	2975
	9400
CLERK	950
MANAGER	2850
SALESMAN	5600

The tricky part of this particular pivoting operation is handling NULLs correctly. For one thing, the JOB subtotals need to be pivoted to the rightmost column, but they have no DEPTNO value to pivot to. For another thing, the input might have NULLs in the JOB or DEPTNO columns, so I need a reliable way to identify the output rows that have subtotals.

I use the GROUPING() function to identify the subtotals:

- When GROUPING(DEPTNO) is equal to 1, the row contains a JOB subtotal (or the grand total) and I have to assign an arbitrary DEPTNO value so I can pivot.
- When GROUPING(JOB) is equal to 1, the row contains a DEPTNO subtotal (or the grand total) so after pivoting I output 'Total' in the JOB column of the last row.
-

```
select case gr_job when 1 then 'Total' else job end job,
  "10", "20", "30", "Total"
from (
  select case grouping(deptno) when 1 then -1 else deptno end
deptno,
    job, grouping(job) gr_job, sum(sal) sal
  from scott.emp
  group by cube(deptno, job)
)
pivot(
  sum(sal) for deptno in (10, 20, 30, -1 as "Total")
)
order by gr_job, job;
```

Contact address:

Stew Ashton
Blog: https://stewashton.wordpress.com/
Twitter: @stewashton