Common Reporting Problems

As a SQL developer, DBA, or power user, at some point you’ll have to write reports to share with colleagues, management, clients, or other companies. If you’re lucky, you’ll be able to concentrate on the logic and presentation of your reports in peace and quiet. Unfortunately, however, you’ll almost certainly be confronted with “shoulder surfing” report critics. You know the kind: they stand behind you, gesticulating at the screen, offering helpful advice like “That should be converted to a spreadsheet.” or “Can you make it look like the old report? It’s easier to read that way.”

Many of these demands will have common solutions, to rotate, compress, substitute and otherwise transform perfectly normal report data into a format that’s more familiar or suitable for the intended audience. This chapter covers many common styles of reporting, and offers some common and not-so-common solutions to help you meet your immediate needs, as well as equip you with ideas to tackle future reporting scenarios we haven’t thought of, but your boss surely has.

11-1. Avoiding Repeating Rows in Reports

Problem

You want to mimic the SQL*Plus BREAK feature to suppress near-identical leading data in rows of data. For instance, in a multi-column report where only the last column differs from one row to the next, you want to blank out the repeated early columns and show only the differing column’s data.

Solution

Oracle’s SQL*Plus tool has supported the BREAK feature for decades, and many basic text reports rely on this crude but effective formatting approach. Unfortunately, the BREAK command isn’t native to SQL, so it can’t be used to achieve the same results outside of SQL*Plus. Using Oracle’s ROW_NUMBER analytic function and the CASE expression, you can mimic the BREAK behavior from any SQL interface.

For those not familiar with the SQL*Plus BREAK feature, the next commands illustrate it “hiding” identical job identifiers and manager identifiers in a simple SELECT statement’s output. Only partial results are shown, but you can clearly see the effect.

```
budget break on job_id on manager_id
```
We can use Oracle’s `CASE` expression and the `ROW_NUMBER` analytic function to mimic the `BREAK` feature from any SQL query interface or programming API. The next SQL statement shows these two features in action, and the identical results generated from any interface.

```sql
select case when job_ctr = 1 then job_id else null end "JOB_ID",
         case when jobman_ctr = 1 then manager_id else null end "MANAGER_ID",
         employee_id
from (select job_id,
               row_number() over (partition by job_id order by job_id) as job_ctr,
               manager_id,
               row_number() over (partition by job_id, manager_id order by job_id, manager_id) as jobman_ctr,
               employee_id
        from hr.employees)
```

As promised, our results are pleasingly similar to the SQL*Plus approach, with precious ink saved in not printing repeating data.