# The case for CASE

By
Mathias Magnusson
November 2009

Table of Contents

# Introduction

In this document we will look closer at the keyword CASE in SQL. While it often is treated just as a readable replacement for decode, it has a few more uses. Readability is a feature in itself, but the most misunderstood part of case among developers is how flexible it is.

This article will only look at the basic use and not try to dig into every aspect of it. Thus it is geared more towards the intermediate SQL user than to the experts.

# Select

The CASE keyword was introduced in 8i (9i for PL*SQL) and essentially replaces DECODE. I confess to never liking decode, though it is powerful, it is also ugly and sometimes close to unreadable. Case introduces a few things decode cannot do, but it makes it more accessible to those who write SQL occasionally. It is easier to get a case formatted nicely than it is to do the same with decode.

The basic syntax for case is:

```
select empno
       ,ename
       ,case
          when sal       < 1500          then 'low'
          when sal between 1500 and 3000 then 'mid'
          when sal       > 2500          then 'high'
       end sal_class
   from emp;
```

Here we have classified the people by their salary. Those earning less than 1500 will be tagged as *low*, those earning 1500-2500 will be *mid*, and those earning more will be *high*. This kind of case construct is called *searched* and is the kind I find most often in SQL. There is however a simpler version that can be used when the comparison is on just one value and against just one value per condition. This has the creative name of *simple case*.

```
select empno
      ,ename
      ,case deptno  when 10 then 'Dept 10'
                    when 20 then 'Dept 20'
                    when 30 then 'Dept 30'
        end dept
  from emp;
```

This basic example just rewrites the value of *deptno* to have “Dept” in front of it. It shows the use of the simple case. It is often forgotten and rewritten as a searched case even when the simple case would do.

One thing to note here is that the else construct that can be used to give a value when a situation not covered in the conditions is encountered is not mandatory in SQL the way it is in PL/SQL. In SQL it defaults to null while it ends up with an exception in PL/SQL. The last example could be enhanced to use else like this:

```
select empno
      ,ename
      ,case deptno when 10 then 'Dept 10'
                   when 20 then 'Dept 20'
                   else         'unknown'
       end
  from emp;
```

You recognize a *simple case* on the expression between the case and when keywords. The expression is the department number (*deptno*). A *searched case* has nothing between case and when.

In this example, department 30 was not know at the time the SQL was written and it is now showing up as ”unkown” when the SQL is executed. Note that the syntax is ”else <value>”. One common mistake is to write it as ”else then <value>”. It is easy enough to fix, but Oracle will not help you more than to inform you that you have a missing expression. The time I’ve lost looking for an error just to find the extra *then* is embarrassing. It looks correct since all the previous lines has it, but the compiler is uncompromising on the requirement to remove it.

One thing to remember is that multiple conditions may be true for a single row. In this case all three conditions are true. So does it stop after the first true value or does it run through all, and the final result is the result from last true condition?

```
select case
          when empno = 7839   then 1
          when ename = 'KING' then 2
          when sal   = 5000   then 3
       end all_true
  from emp
 where empno = 7839;
```

The result is 1 and the reason is that it takes the first true value and then leaves the case statement. The above statement also shows that a searched case does not need to have any correlation between the different conditions. This is one of the strengths of case, while simultaneously being one of the most underused features.

# Update

All the power of case in the select clause can also be used in other parts of the SQL language. For example, it is very common to see an update script that looks something like this.

```
update emp
   set sal = sal * 1.15
 where deptno = 10;

update emp
   set sal = sal * 1.05
 where deptno = 20;

update emp
   set sal = sal * 1.10
 where deptno = 30;
```

This is of course not much of a problem in a table with 14 rows, but say that each statement takes an hour or more to complete and updates just a small portion of the whole table. The time to run each statement can then be a problem if the conditions are such that the whole table has to be read each time. The above statements can be rewritten like this.

```
update emp
    set sal = sal * (case when deptno = 10 then 1.15
                          when deptno = 20 then 1.05
                          when deptno = 30 then 1.10
                     end);
```

*Case* is here used to return the factor to use to adjust everyones salary. I find this easier to read and if there are many update statements it will give a better overview over what changes are intended to be made.

Another situation where case is useful is when there are many columns than needs to be updated, but the condition is not the same for each update.

```
update emp
   set sal = sal * (case deptno
                      when 10 then 1.15
                      when 20 then 1.05
                      when 30 then 1.10
                    end)
      ,comm = (case job
                 when 'PRESIDENT' then 1500
                 when 'MANAGER'   then 1000
                 when 'SALESMAN'  then 3000
                 when 'ANALYST'   then  500
                 else                   200
               end);
```

Solving this with individual update statements would require eight individual statements if we assume that there could be more departments and more job titles than specifically listed here. There could also be more complicated situations causing more complex situations.

# Where

Using case in a where is often useful when there are multiple combinations that all needs to be treated as true. This can be especially true if some additional tests should be performed for all such conditions. Consider this example.

```
select *
  from emp
 where case
         when job = 'PRESIDENT' and sal > 4500 then 1
         when job = 'MANAGER'   and sal > 2500 then 1
         when job = 'SALESMAN'  and sal > 1500 then 1
         else                                       0
       end = 1
   and sal / 10 = trunc(sal/10)
   and ename like '%A%';
```

Here we will retrieve employees that has a salary above a certain threshold per title for just some of the titles an employee can have. For all those we want to make sure the salary can be divided by ten without getting a rest and their name should contain an A. This example may not be very realistic, but it shows the power of using case in the where clause. Note how the condition on the case compares the result of the whole case expression with a value. It may look very strange in the beginning, but it is very useful once you get used to that syntax.

# Order By

Sometimes sorting needs to be done in a way that neither descending nor ascending will solve. Say that you want to sort the employees such that those in department 20 is listed before those in 10, with those in department 30 coming last, then something like this would do the trick.

```
select *
  from emp
 order by case deptno
            when 10 then 2
            when 20 then 1
            when 30 then 3
            else         4
          end;
```

This translates 20 to 1, 10 to 2, 30 to 3, and anything else to 4 and those translated values is then what the rows is sorted by. This can be very powerful when specialized reporting needs has to be supported.

Another case is when you want to treat two groups as if they were one. Say that you want to return all employees ordered by deptno and salary, but department 10 and 30 should be treated as the same. That is employees within those departments should be grouped together and ordered by salary. This SQL shows one way to achieve that.

```
select *
  from emp
 order by case deptno
            when 10 then 10
            when 30 then 10
            else         deptno
          end
         ,sal desc;
```

Note how all other departments just keeps their deptno via the else. In fact, the ”when 10” line could be removed as it doesn’t change the value. I would keep it just for the declarative value in showing exactly what

we want to achieve.

## Group By

Using case when aggregating data can provide some unexpected power. Let us start with a *simple case* where we report total salary for management, workers, and sales.

```
select case job
         when 'PRESIDENT' then 'MGMT'
         when 'MANAGER'   then 'MGMT'
         when 'SALESMAN'  then 'SALES'
         when 'CLERK'     then 'WORK'
         when 'ANALYST'   then 'WORK'
       end cat
      ,sum(sal)
  from emp
 group by case job
            when 'PRESIDENT' then 'MGMT'
            when 'MANAGER'   then 'MGMT'
            when 'SALESMAN'  then 'SALES'
            when 'CLERK'     then 'WORK'
            when 'ANALYST'   then 'WORK'
          end;
```

The same case statement is used twice here. First to translate the job titles to the categories we are interested in and then to group by the same categories. The one difference is that a column alias can not be declared for it in the group by (see *cat* in select clause).

It would of course be possible to also use case to control the final order of the report or having to filter the aggregated data. Here is an example of doing both in an expanded version of the previous example.

```
select case job
         when 'PRESIDENT' then 'MGMT'
         when 'MANAGER'   then 'MGMT'
         when 'SALESMAN'  then 'SALES'
         when 'CLERK'     then 'WORK'
         when 'ANALYST'   then 'WORK'
         end cat
      ,sum(sal)
  from emp
 group by case job
            when 'PRESIDENT' then 'MGMT'
            when 'MANAGER'   then 'MGMT'
            when 'SALESMAN'  then 'SALES'
            when 'CLERK'     then 'WORK'
            when 'ANALYST'   then 'WORK'
          end
having case
         when sum(sal) <  9000 then 1
         when sum(sal) > 11000 then 1
         else                       0
       end = 1
order by case cat
           when 'PRESIDENT' then 3
           when 'MANAGER'   then 3
           when 'SALESMAN'  then 1
```

```
    when 'CLERK'     then 2
    when 'ANALYST'   then 2
end;
```

Is this an excessive use of *case*? Probably, and it could be more maintainable by using subquery refactoring (WITH clause) or an inline view (select in the from clause). That would allow the same functionality without repeating the same thing so many times. However, this article's focus is to show *case* and adding in more SQL features would not help understanding the powerful features of case.

## Pivot

Say that you want to produce a report showing each department on a row with a total salary per title. It can be done with a series of unions and summing the results, but it can be done with a single select using case. Here is one version of this.

```
select deptno
      ,sum(case job when 'PRESIDENT' then sal else 0 end) pres_sal
      ,sum(case job when 'MANAGER'   then sal else 0 end) mgr_sal
      ,sum(case job when 'SALESMAN'  then sal else 0 end) sls_sal
      ,sum(case job when 'CLERK'     then sal else 0 end) clk_sal
      ,sum(case job when 'ANALYST'   then sal else 0 end) anl_sal
 from emp
group by deptno;
```

We group the data by *deptno* and use one case per title to report just salaries for that title in that column. This gives us a nice report showing how much salary each department spends in each job title.

This is a very powerful feature once you wrap your head around the basic construct. You will start recognizing areas where this technique can be used. You will also be able to stop co-workers from reinventing the wheel when a pivot need comes up.

# Conclusion

This article has reviewed the use of CASE in SQL. It is very powerful and this article has tried to show some of the many flexible ways it can be used to extend the power of SQL.

I hope this article helps you. Please send comments and questions to mathias.magnusson@gmail.com or visit the comment page linked to from http://mathiasmagnusson.com/oracle-talk/articles.