Oracle Analytic functions

***Function(arg1,..., argn) OVER ( [PARTITION BY <...>] [ORDER BY <....>] [<window\_clause>] )***

Oracle analytic functions are similar to aggregate or group by functions, except they do not group the entire result set.

Consider this query:

**select name, count(\*) from student\_course**

**group by name**

It will return each person from student\_course with the number of entries.

Now, suppose you want each student record with a count of how many courses they have registered for.

In normal SQL you would do something like this:

**Select a.name, a.course\_identification, b.course\_count**

**From student\_course a,**

**(select name, count(\*) as course\_count from student\_course**

**group by name) b**

**where a.name = b.name**

It uses two queries to get the results you need.

With analytic functions, you can create one query to achieve the same results:

**Select a.name, a.course\_identification,**

**count(\*) over (partition by name) as course\_count**

**From student\_course a**

Partition by works just like the group by, so we are saying we want the count for each student.

In most cases analytic functions will run as fast as or faster than other methods.

So, when you want to use aggregate functions but still need each row of detail, analytic functions are the way to go.

## But wait!!! There is more….

Oracle analytical functions go much deeper than this:

Supposing you wanted to list the first course a student registered for with their record.

This would take quite a bit of SQL to do normally, however it is a breeze with analytic functions.

First, let’s identify the problem:

We want to get the first course number by the start date and add it to the student course results.

**Select a.name, a.course\_identification,**

**first\_value(a.course\_identification) over (partition by name order by start\_date) as first\_course**

**From student\_course a**

What’s going on here then?

The first thing to look at is the partition by:

We are grouping by each student.

Next look at the order by:

We are ordering by the start\_date of each course

Finally, look at the function:

It is taking the first value of course identification.

To put it all together, we end up with:

We get a list of course identifications for each student, and we order them by the start\_date, then we take the first one in that list.

## But wait!!! There is even more…

Let’s say we want to show the previous course (if any) that a student registered for..

We can achieve this by using Lag or Lead. They both follow this syntax:

***LEAD (<sql\_expr>, <offset>, <default>) OVER (<analytic\_clause>)***

Lag and Lead can give us the next or previous row in a clause (the partition by/order by bits).

It’s easier to see in action than explain it:

**Select a.name, a.course\_identification,**

**LAG(a.course\_identification,1,’NONE’) over (partition by name order by start\_date) as first\_Previous\_Course**

**From student\_course a**

So, as in previous examples, we are getting a list of course ids for each person, and ordering them by start\_date.

The LAG is saying get the course id which is 1 row before the current course id, and if there is no row, then show “NONE”.

## But wait!! There is even more than all of that!!!

Analytic functions can get even more complex by adding ROW or RANGE windows.

The general syntax is:

***[ROW or RANGE] BETWEEN <start\_expr> AND <end\_expr>***

This is a rather complex concept, but makes a lot of sense in certain situations:

**Select a.name, a.course\_identification,**

**count(\*) over (partition by name order by start\_date range between start\_date and add\_months(start\_date,-3)) as course\_count\_last\_3M**

**From student\_course a**

This query will get us the count of courses that a student registered for in the last 3 months.

References:

<http://docs.oracle.com/cd/E11882_01/server.112/e26088/functions004.htm>

<http://psoug.org/reference/analytic_functions.html>