Table of Contents

# 1. GREATEST and LEAST

**GREATEST** and **LEAST** return the largest and smallest value from the list of arguments. Notice what happens with nulls. If there is a null in the list then the functions treat this as an unknown value and therefore the function cannot "know" which value in the list is the largest.

Warning: in a future unit we discuss the functions Max and Min which sound very much like Greatest and Least. They are NOT the same. Greatest and Least are row functions**- they work on multiple columns in a single row of data**. You can use them with literals. Max and Min work down a column over a set of rows.

Demo 01: Greatest, Least.

```
select A
, GREATEST(B,C,D,E,F)
, LEAST(B, C, D, E,F)
from z_tst_numbers;
```

```
       A GREATEST(B,C,D,E,F)  LEAST(B,C,D,E,F)
-------- -------------------  ----------------
       1                  90               10
       2
       3
       4
       5                   0                0
       6                  10               10
       7                  85             -210
       8                  85             -210
       9                 200               -1
      10                 200               -1
```

Data type Issues: A function returns a single value. GREATEST (list), LEAST (list) returns the largest, smallest in the list. If the list is of mixed data type, the data type of the first argument is used and other elements of the list are converted to that data type. You may wish to use a conversion function to force a specific data type for the first argument.

Demo 02: greatest, least with numbers

```
select
  GREATEST(4, 45.78, 9, -333.98)
, LEAST   (4, 45.78, 9, -333.98)
from dual;
```

```
GREATEST(4,45.78,9,-333.98) LEAST(4,45.78,9,-333.98)
--------------------------- ------------------------
                      45.78                  -333.98
```

Demo 03: greatest, least with strings; the first argument is a string, so all of the other arguments are cast to strings. Note that the least function returns the string '4', not the number 4.

```
  select
    greatest( 'flea', 4, 45.78, 9, 'elephant', 9)
  , least   ( 'flea', 4, 45.78, 9, 'elephant', 9)
  from dual;
```

```
GREA L
---- -
flea 4
```

Demo 04: Data types matter; as numbers 25 is smaller than 125; as string '125' sorts before '25' and is considered least.

```
  select least(25, 125) as Numbers, least('25', '125') as Strings
  from dual;
```

```
   NUMBERS STR
---------- ---
        25 125
```

Demo 05: This has a first argument that is numeric; when the function gets to the argument 'elephant' that cannot be cast to a number and we get an error.

```
  select greatest(4, 45.78, 9, 'elephant') from dual;
```

```
select greatest(4, 45.78, 9, 'elephant') from dual
                              *
ERROR at line 1:
ORA-01722: invalid number
```

## 1.1. Examples using the demo tables

Demo 06: This query returns the largest of the two columns quoted_price and  prod_list_price

```
  select order_id
  , prod_id
  , quoted_price
  , prod_list_price
  , GREATEST (quoted_price, prod_list_price)  as HigherPrice
  from oe_orderDetails
  join prd_products using (prod_id)
  where prod_id in (1000, 1010);
```

```
    ORDER_ID    PROD_ID QUOTED_PRICE PROD_LIST_PRICE HIGHERPRICE
------------ ---------- ------------ --------------- -----------
         528       1010          150             150         150
         390       1010          175             150         175
         395       1010          195             150         195
         303       1000          125             125         125
         313       1000          125             125         125
         550       1010          175             150         175
         551       1010          175             150         175
         605       1010          125             150         150
         608       1000          100             125         125
         609       1010          175             150         175
         105       1010          150             150         150
         405       1010          150             150         150
         115       1000          100             125         125
        2120       1010          175             150         175
        2121       1010          175             150         175
```

```
      3808       1000           100              125         125
      3808       1010           125              150         150

17 rows selected.
```

Demo 07: The Greatest function just returns a number. You probably want to know which is bigger; that uses a Case. (In our tables these price columns are not null, but that is not true for all tables. So you always have to think about those nulls.)

```
select order_id
, prod_id
, quoted_price as "quoted"
, prod_list_price  as "List"
, GREATEST (quoted_price, prod_list_price)  as "higher"
, case  when quoted_price = prod_list_price then 'same price'
        when quoted_price > prod_list_price then 'quoted is higher'
        when quoted_price < prod_list_price then 'list is higher'
        else 'one or more is null'
        end  "PriceComparison"
from oe_orderDetails
join prd_products using (prod_id)
where prod_id in (1000, 1010)
;
```

```
    ORDER_ID    PROD_ID     quoted       List     higher PriceComparison
------------ ---------- ---------- ---------- ---------- -------------------
         528       1010        150        150        150 same price
         390       1010        175        150        175 quoted is higher
         395       1010        195        150        195 quoted is higher
         303       1000        125        125        125 same price
         313       1000        125        125        125 same price
         550       1010        175        150        175 quoted is higher
         551       1010        175        150        175 quoted is higher
         605       1010        125        150        150 list is higher
         608       1000        100        125        125 list is higher
         609       1010        175        150        175 quoted is higher
         105       1010        150        150        150 same price
         405       1010        150        150        150 same price
         115       1000        100        125        125 list is higher
        2120       1010        175        150        175 quoted is higher
        2121       1010        175        150        175 quoted is higher
        3808       1000        100        125        125 list is higher
        3808       1010        125        150        150 list is higher
```

## 2. DECODE

The Decode function is included here for completeness and because you may find it in old code and need to figure it out. It is generally a good idea to avoid this in new code. (I do not like it when more than half of this document is taken up by a technique that I do not even want you to use. But Decode exists and is not obvious. I do not have any queries in the assignments or exams where Decode is the appropriate choice.)

The Decode function is used to provide an If-Then-Else logic within a SQL statement. The first argument is an expression to be evaluated. Then you have pairs of arguments; the first of the pair is a possible value for the expression, the second of the pair is the return value. The last, optional, argument is a value to return if the expression is not matched by any of the pairs. Decode is limited to exact matches.

The Decode function can be difficult to debug if it is nested or if the pairings are not obvious. But it is a very common function in older Oracle code. Decode is an Oracle specific function. Use Case to produce more portable code.

Demo 08: We want to give customers a 5% savings for each pet supply item, 5% for each sporting goods item and 10% for each appliance. These savings are taken against the list price

```
  select catg_id, prod_id, prod_list_price
  ,  DECODE (catg_id,
             'PET', 0.95,
             'SPG', 0.95,
             'APL', 0.90,
                    1.00 ) * prod_list_price AS "Today's Price"
  from prd_products;
```

```
--- selected rows shown

CATG_I    PROD_ID PROD_LIST_PRICE Today's Price
------ ---------- --------------- -------------
HW           1000             125           125
SPG          1010             150         142.5
SPG          1020           12.95       12.3025
SPG          1030           29.95       28.4525
HW           1072            25.5          25.5
HW           1080              25            25
HW           1090          149.99        149.99
HW           1100           49.99         49.99
PET          1140           14.99       14.2405
PET          1141           99.99       94.9905
PET          1142             2.5         2.375
```

Demo 09: Use Decode to interpret small collections of coded values that seldom change.

```
  select Order_status
  ,  Decode(order_status,
           0, 'New',
           1, 'Producted',
           2, 'Picked',
           3, 'Shipped',
           9, 'BackOrdered',
              'Invalid Code') AS Order_Status
  from oe_orderHeaders;
```

```
--- selected rows shown

ORD_STATUS ORDER_STATUS
---------- ------------
         9 BackOrdered
         9 BackOrdered
         9 BackOrdered
         2 Picked
         2 Picked
         9 BackOrdered
         3 Shipped
         3 Shipped
         4 Invalid Code
         7 Invalid Code
. . .  rows omitted
```

/* using case*/

```
  select
    Order_status
  , case order_status
    when 0 then 'New'
    when 1 then 'Producted'
    when 2 then 'Picked'
    when 3 then 'Shipped'
    when 9 then 'BackOrdered'
    else 'Invalid Code'
    end  As Order_Status
  From oe_orderHeaders
  ;
```

If this type of descriptive term is subject to change then it is a better idea to put these into a table and do a join to get the descriptive values ( such as we do with the prd_categories table). Suppose you used the previous decode function in a series of queries that were then built into application programs. Five years later the company decides to use ord_status 4 and has a description for it- or decides that the label 'Producted' should be replaced with another term. Now someone would have to find all of those queries that were embedded in all of those application programs that use this decode function and change them and test them and update all of the applications.

# 3. Things to watch out for with Decode

## 3.1. Missing else value

Demo 10: This is the first decode demo but I have skipped the "else" value. That that case Decode returns a null and it looks like a lot of products do not have a price value for the last column. You do not want to do this. The business rule was to give a discount for certain types of items; in a business situation that would not mean that the price of HW item. was unknown

You should be aware when a function, such as Decode, will produce a null as output but you should try to avoid those situations.

```
  select catg_id, prod_id, prod_list_price
  , DECODE (catg_id,
            'PET', 0.95,
            'SPG', 0.95,
            'APL', 0.90 ) * prod_list_price AS "Today's Price"
  from prd_products;
```

```
--- selected rows shown

CATG_ID PROD_ID                PROD_LIST_PRICE        Today's Price
------- ---------------------- ---------------------- ----------------------
HW      1000                   125
SPG     1010                   150                    142.5
SPG     1020                   12.95                  12.3025
SPG     1030                   29.95                  28.4525
HW      1072                   25.5
HW      1080                   25
HW      1090                   149.99
HW      1100                   49.99
PET     1140                   14.99                  14.2405
PET     1141                   99.99                  94.9905
PET     1142                   2.5                    2.375
```

## 3.2. Return type problems

The return value type is determined by the first return value in the Decode.

Demo 11: The following will run, the return type will be varchar2 and the numeric value 345 will be cast to the string '345'

```
  select
    Order_status
  , DECODE(order_status,
    0, 'New',
    1, 'Producted',
    2, 'Picked',
    3, 'Shipped',
    9, 345,
    'Invalid Code') As OrderStatus
  from oe_orderHeaders
  Order By order_status
```

```
--- selected rows shown

ORD_STATUS             ORDER_STATUS
---------------------- ------------
1                      Producted
1                      Producted
2                      Picked
2                      Picked
2                      Picked
3                      Shipped
3                      Shipped
4                      Invalid Code
4                      Invalid Code
5                      Invalid Code
5                      Invalid Code
7                      Invalid Code
7                      Invalid Code
9                      345
9                      345
```

Demo 12: This will not run. The return type is determined by the first possible return value ( in this query 150) which is numeric and the other values such as "Producted" cannot be cast to a number

```
  select Order_status
  , Decode(order_status,
          0, 150,
          1, 'Producted',
          2, 'Picked',
          3, 'Shipped',
          9, 345,
             'Invalid Code') AS OrderStatus
  from oe_orderHeaders
  order by order_status
  ;
```

```
SQL Error: ORA-01722: invalid number
```

Where this can get very confusing is when you have a Decode that sometimes works and sometimes does not. Suppose we had the following decode expression where status is some numeric column in the table.

```
Decode(order_status,
       0, 150,
       1, 250,
       2, 350,
       9, 'Invalid status',
          'Invalid Code')
```

If the table contains only the values 0, 1, 2 in the status column this will work. But if a new row is added with a value for status other than 0, 1, or 2 then the query will fail. You need to write queries that will not fail for any legitimate data in the table.

You can try this with a CTE that gets only rows with order_status 0,1,and 2. and then use those rows in the main query with the above decode.

```
With tbl as (
  select *
  from oe_orderHeaders
  where order_status in ( 0,1,2)
  )
 select distinct
   order_status
 , Decode(order_status,
      0, 150,
      1, 250,
      2, 350,
      9, 'Invalid status',
         'Invalid Code')  as DecodeVal
 from tbl;
```

```
ORD_STATUS  DECODEVAL
---------- ----------
         1        250
         2        350
```

But if you change the cte to allow other values, then the query fails. Change the IN list in the CTE to where ord_status in ( 0,1,2, 3,4 ) and then the query produces an error message.

```
         'Invalid Code')  as DecodeVal
         *
ERROR at line 13:
ORA-01722: invalid number
```

The problem is that the decode takes its data type to return from the first pair- which in this case is a number(150). When the query gets to a 3 or 4, then it wants to return 'Invalid Code' which cannot be cast to a number.

### 3.3. Old code styles

You would not normally be writing Decode in queries you write today but you might have to maintain old SQL and you might come across very complex logic done with Decode. This is a very simple example, but it might not be obvious.

Demo 13: Nested decode using the Sign function

```
select
  catg_id
, prod_id
, prod_list_price
, decode(catg_id,
     'PET',
     decode(sign(prod_list_price - 10),
        -1, 'LowCost pet item',
            'HighCost pet item'
        ),
     'SPG',
     decode(sign(prod_list_price - 25),
        -1,  'LowCost sports item',
              decode(sign(prod_list_price - 150),
                 -1,  'MidCost sports item',
                  0,  'MidCost sports item',
                  1 , 'HighCost sports item'
                 )
        ),
     'APL', 'appliance item'
```

```
        )  AS "Result"
   from prd_products
   where prod_list_price <=
      decode (catg_id,
          'PET', 100,
          'SPG', 300,
          'APL', 400,
          'MUS',  10,
                  75)
   order by catg_id, prod_list_price
   ;
```

```
CATG_ID PROD_ID                PROD_LIST_PRICE        Result
------- ---------------------- ---------------------- --------------------
APL     1130                   149.99                 appliance item
APL     4569                   349.95                 appliance item
GFD     5001                   5
GFD     5000                   12.5
HD      5008                   12.5
HD      5004                   15
HD      5002                   23
HD      5005                   45
HW      1100                   49.99
HW      1110                   49.99
MUS     2487                   9.45
MUS     2412                   9.87
PET     1143                   2.5                    LowCost pet item
PET     1142                   2.5                    LowCost pet item
PET     1150                   4.99                   LowCost pet item
PET     1140                   14.99                  HighCost pet item
PET     1151                   14.99                  HighCost pet item
SPG     1020                   12.95                  LowCost sports item
SPG     1030                   29.95                  MidCost sports item
SPG     1010                   150                    MidCost sports item
SPG     1060                   255.95                 HighCost sports item
SPG     1050                   269.95                 HighCost sports item
```

If you see nested decodes in a query, you need to investigate it carefully. The use of the sign function in a decode is generally a technique for checking if a number is equal to,  less than  or greater than  a specific number.

The use of case expressions here is easier to use and easier to maintain.