**Benefits of Using PRAGMA UDF**

Sources: <https://mwidlake.wordpress.com/2015/11/04/pragma-udf-speeding-up-your-plsql-functions-called-from-sql/>

**A new feature for PL/SQL was introduced in V12, pragma UDF. UDF stands for User Defined Functions. It can speed up any SQL you have that uses PL/SQL functions you created yourself.**

{please see [this second post on some limitations of pragma UDF](https://mwidlake.wordpress.com/2015/11/11/pragma-udf-some-current-limitations/) in respect of IN & RETURN data types and parameter defaults}.

We can create our own functions in PL/SQL and they can be called from both PL/SQL **and** SQL. This has been possible since V7.3 and is used extensively by some sites to extend the capabilities of the database and encapsulate business logic.

A problem with this, though, is that every time you swap from SQL to PL/SQL (or the other way around) you have to do a *context switch*each time, which can be quite cpu and memory intensive. If you are using your own PL/SQL function in the SELECT list of a SQL statement and you are selecting a lot of rows (say as part of a business report) then the overhead can be quite considerable as you *could be doing a context switch per row*. I won’t go into too much detail here (partly as I go in to considerable detail on the subject in a book I am contributing to for 2016 {and is now out – look to the right!}) on how you can investigate the context switching and when exactly it occurs, but I will show you one of the two new ways in Oracle 12 to reduce the overhead, namely PRAGMA UDF. At present this seems to be a little used and rarely-mentioned feature on the blogsphere, with articles just covering simple examples of almost no-business-function, numeric functions.

I’ll give you a slightly less simple example but my next post will give you details on some limitations of pragma UDF. Here I am just setting the scene. I have the below PERSON table which has the parts of the names in distinct columns, with the contents forced to upper case (as is standard practice). We will create a function to provide a nicely init-capped and spaced display name and a second function which is identical but uses PRAGMA UDF.

|  |
| --- |
| PERSON  Name                                     Null?    Type  ---------------------------------------- -------- ---------------  PERS\_ID                                  NOT NULL NUMBER(8)  SURNAME                                  NOT NULL VARCHAR2(30)  FIRST\_FORENAME                           NOT NULL VARCHAR2(30)  SECOND\_FORENAME                                   VARCHAR2(30)  PERS\_TITLE                                        VARCHAR2(10)  SEX\_IND                                  NOT NULL CHAR(1)  DOB                                               DATE  ADDR\_ID                                           NUMBER(8)  STAFF\_IND                                         CHAR(1)  LAST\_CONTACT\_ID                                   NUMBER(8)  PERS\_COMMENT                                      VARCHAR2(2000)    create or replace function normal\_disp\_name (p\_sn      in varchar2                                              ,p\_fn1     in varchar2                                              ,p\_fn2     in varchar2                                              ,p\_title   in varchar2  ) return varchar2 is  v\_return     varchar2(1000);  begin    v\_return := case when p\_title is null then ''                     else initcap(p\_title)||' '                end              ||initcap(p\_fn1)||' '              ||case when p\_fn2 is null then ''                     else substr(p\_fn2,1,1)||' '                end              ||initcap(p\_sn);  return v\_return;  end;  /  create or replace function udf\_disp\_name (p\_sn      in varchar2                                           ,p\_fn1     in varchar2                                           ,p\_fn2     in varchar2                                           ,p\_title   in varchar2  ) return varchar2 is  -- The Below is the KEY bit  PRAGMA UDF;  v\_return     varchar2(1000);  -- {Identical to normal\_disp\_name from here}  -- demo of the code  -- select some data with one of the functions, it does not matter which  select pers\_title title,    first\_forename    ,second\_forename    , surname        ,normal\_disp\_name(p\_sn =>surname           ,p\_fn1  =>first\_forename                         ,p\_fn2=>second\_forename   ,p\_title=>pers\_title) display\_name  from person  ...    TITLE   first\_fn   secon\_fn   SURNAME         DISPLAY\_NAME  ------- ---------- ---------- --------------- ----------------------------  MR      HARRISON   RICHARD    HARRIS          Mr Harrison R Harris  MRS     ANNEKA     RACHAEL    HARRIS          Mrs Anneka R Harris  MRS     NICKIE     ALISON     ELWIG           Mrs Nickie A Elwig  MASTER  JAMES      DENZIL     ELWIG           Master James D Elwig  MR      JEFF                  GARCIA          Mr Jeff Garcia  ...  MRS     AMELIA     MARIA      ORPINGTON-SMYTH Mrs Amelia M Orpington-Smyth |

So we have our test table, you can see my normal\_disp\_name function and that the \*only\* difference with the second version, udf\_disp\_name, is the inclusion of PRAGMA\_UDF in the declaration section. That is partly why it is such a nice feature, you can just add this one line to existing code and you should get the benefit. Should….. {see second post when I do it}

Finally, I show some code using the normal\_disp\_namefunction and the output.

To demonstrate the impact of context switching I will select 100,000 records from my test table in 3 ways: using only native SQL functions and thus no context switching; using my traditional PL/SQL function which suffers from context switching; with my new “pragma UDF” function to reduce the overhead of the context switching.

|  |
| --- |
| select avg(length(        case when pers\_title is null then ''                     else initcap(pers\_title)||' '                end              ||initcap(first\_forename)||' '              ||case when second\_forename is null then ''                     else substr(second\_forename,1,1)||' '                end              ||initcap(surname)            )      )  avg\_name\_length         ,count(\*)  from person  where pers\_id > 100100  and rownum < 100000    select  avg(length(normal\_disp\_name(p\_sn =>surname           ,p\_fn1  =>first\_forename                                     ,p\_fn2=>second\_forename   ,p\_title=>pers\_title)        ) ) disp\_name\_len         ,avg(addr\_id)         ,count(\*)  from person  where pers\_id > 100100 and rownum < 100000    select  avg(length(udf\_disp\_name(p\_sn =>surname           ,p\_fn1  =>first\_forename                                  ,p\_fn2=>second\_forename   ,p\_title=>pers\_title)        ) ) disp\_name\_len         ,avg(addr\_id)         ,count(\*)  from person  where pers\_id > 100100 and rownum < 100000 |

One thing to mention in passing is that the code using either function is much easier to read and is self-documenting. These are a couple of the benefits of proceduralising your code, as well as creating just one place to maintain it. If I had to re-use that native-SQL section in a half-dozen reports I would probably mess up at least one of the times I cut-and-pasted it and I would now have several places to maintain that code.

I ran the test statements several times and took the average of the 2nd to 6th runs, to remove the initial parsing & caching overhead that comes with the first execution and to get more reliable figures than one further run would give me.

|  |
| --- |
| Version                      Run Time average (secs)  Native SQL                   0.03  Traditional PL/SQL           0.33  PRAGMA UDF PL/SQL            0.08 |

As you can see, just including PRAGMA UDF removed most of the overhead caused by context switching.

How does PRAGMA UDF work? I’m not sure, the official Oracle documentation is pretty light on it and just says:

“The UDF pragma tells the compiler that the PL/SQL unit is a user defined function that is used primarily in SQL statements, *which might improve its performance*”

Note the italics (which are mine). “Might improve its performance” but no detail as to what it does. As I understand it, the ‘C’ calls that are required when processing the data within PL/SQL are reduced and SQL\*Plus may be batching the calls – but treat that as a bit of wild speculation for now.

Pragma UDF can slow down slightly functions being called directly from PL/SQL. So use it only for functions you know will be called from SQL.

I’ll make one other observation. Using PL/SQL functions increased the run time to process 100,000 records on my modest test system by all of 0.3 seconds. But that is 10 times the time taken for the native SQL statement. Pragma UDF removes around 80% of this overhead. It’s a nice saving but is probably inconsequential if your code is actually doing any physical IO at all (my example is processing already cached blocks). And if you are only processing a few records or one record in a GUI screen, the context switching is moot {meaning, of no significance}.

But if you have code that processes a huge set of data and uses a lot of user defined PL/SQL functions (and again I go into a lot more detail about this in the book) using pragma UDF in 12C could gain you quite a bit of extra performance. If you have code where even 0.00001 seconds is important (think trading systems) then again there may be a worthwhile benefit.

I should also make it clear that the context switch happens when you use built-in PL/SQL functions. But of course you can’t add the pragma UDF to Oracle’s code. (Well, you might be able to but you should not!)

**Pragma UDF – Some Current Limitation Using**

here are currently some limitations to when pragma UDF will speed up your calls to PL/SQL functions from SQL.

In my [post introducing the new pragma UDF feature](https://mwidlake.wordpress.com/2015/11/04/pragma-udf-speeding-up-your-plsql-functions-called-from-sql/) of Oracle 12c I explained how it can be used to reduce the impact of context switching when you call a PL/SQL function from SQL.

In my example I showed how running a SQL-only SELECT statement that formatted a name for display over 100,000 records took 0.03 seconds went up to 0.33 seconds when the formatting SQL was put in a user defined PL/SQL function. This impact on performance is a shame as it is so beneficial to encapsulate business logic in one single place with PL/SQL. Stating that the PL/SQL function is a user defined one with the pragma UDF option reduced the run time to 0.08 seconds – which is removing most of the context switching overhead. Check out the prior post for full details.

This improvement in performance is great news and is as good, and sometimes better, than using the other new capability of 12c – allowing you to state a function as part of a SQL statement using the WITH clause, if you know about that (I plan to do a further post on that). As a quick example, here is my display name function code expressed within a WITH clause:

|  |
| --- |
| with    function l\_disp\_name(p\_sn      in varchar2                        ,p\_fn1     in varchar2                        ,p\_fn2     in varchar2 :=null                        ,p\_title   in varchar2 :=null )  return varchar2  is  v\_return     varchar2(1000);  begin    v\_return := case when p\_title is null then ''                     else initcap(p\_title)||' '                end              ||initcap(p\_fn1)||' '              ||case when p\_fn2 is null then ''                     else substr(p\_fn2,1,1)||' '                end              ||initcap(p\_sn);  return v\_return;  end l\_disp\_name;  select  max(l\_disp\_name(p\_sn =>surname           ,p\_fn1  =>first\_forename                         ,p\_fn2=>second\_forename   ,p\_title=>pers\_title)        ) text\_output         ,count(\*)  from pers  / |

The above runs in 0.10 seconds, just slightly slower than 0.08 for my pragma UDF function

However, I need to warn you of some current limitations to pragma UDF. Nearly all the examples on the web so far are

* using very, very simple functions that take in a number and return a number
* Use a stand-alone stored function

And they work fine. However, I had real trouble getting a performance gain when I was working with my function that took in four varchar2 inputs and returned a varchar2 value. No error was given when I marked the function with pragma UDF but there was no performance gain (or loss).

I eventually worked out some limitations to pragma UDF on my version of Oracle – 12.1.0.2.0

1. It gives a performance boost when the inputs and return values are NUMBER, VARCHAR2, multiple VARCHAR2 IN parameters
2. There is no performance boost when either or both the IN parameter or RETURN value is a DATE
3. There is no performance boost if there are any default values for VARCHAR2 IN parameters
4. If the function gains a performance benefit from pragma UDF as a standalone stored function, it appears to also gain an advantage if it is a function defined as pragma UDF within a package – so you can still keep all your functions in packages.

You might notice that my example of using the WITH clause states a function that has default values. The WITH option gains the performance advantage of that feature just fine with IN parameter defaults.

The take-home message is that, at present, pragma UDF only seems to help functions with certain types of IN or RETURN values and is nullified by default values – so if you see no performance gain for your functions, this might be why. I need to stress that my tests were not exhaustive, I have not investigated many other combinations.

I’ve discussed the issue with a couple of people within Oracle and the relevant Product Manager is looking to investigate further for me, which is jolly decent of the fellow.

My investigation is of course only by empirical testing, it does not reveal how pragma UDF works. But, as I said in my first post, it seems to aid how information is passed between the PL/SQL and SQL engines as it is variation in those that seem to nullify the benefit of pragma UDF. If you want to duplicate my tests, you can do with the below scripts. I show my test output first, with comments produced with PROMPT commands. I then give you the SQL to create the test table, the functions and package I used and the test script. Feel free to take, expand and let me know of anything different or further you may find. I say nothing of interest after the scripts, so this is in effect the end of the post

The output of my test, with prompts:

|  |
| --- |
| running udf\_tests  investigating why pragma udf helps some simple functions and not others  ---------------------------------------------------------------------- --    simple number in-number out function  NUMBER\_OUTPUT   COUNT(\*)  ------------- ----------    10000000000     100000  Elapsed: 00:00:00.12    NUMBER\_OUTPUT   COUNT(\*)  ------------- ----------    10000000000     100000  Elapsed: 00:00:00.03  \*\* udf helps    simple varchar in-varchar out function  TEXT\_OUTPUT                      COUNT(\*)  ------------------------------ ----------  XYYYY                              100000  Elapsed: 00:00:00.12    TEXT\_OUTPUT                      COUNT(\*)  ------------------------------ ----------  XYYYY                              100000  Elapsed: 00:00:00.04  \*\* udf helps    two varchar in-varchar out function, is the issue with more than one in parameter?  TEXT\_OUTPUT                      COUNT(\*)  ------------------------------ ----------  XYYYYYYYYY                         100000  Elapsed: 00:00:00.14    TEXT\_OUTPUT                      COUNT(\*)  ------------------------------ ----------  XYYYYYYYYY                         100000  Elapsed: 00:00:00.04  \*\* udf helps    simple date in-date out function  DATE\_OUTPUT            COUNT(\*)  -------------------- ----------  14-MAY-2010 13:11        100000  Elapsed: 00:00:00.15    DATE\_OUTPUT            COUNT(\*)  -------------------- ----------  21-NOV-2004 13:11        100000  Elapsed: 00:00:00.15  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SIMILAR TIME!!!  udf does not help    is date out the issue,  date in-num out function  NUMBER\_OUTPUT   COUNT(\*)  ------------- ----------        2454431     100000  Elapsed: 00:00:00.17    NUMBER\_OUTPUT   COUNT(\*)  ------------- ----------        2454231     100000  Elapsed: 00:00:00.18  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SIMILAR TIME!!!  udf does not help    is date in the issue,  num in-date out function  DATE\_OUTPUT            COUNT(\*)  -------------------- ----------  07-AUG-2018 18:11        100000  Elapsed: 00:00:00.21    DATE\_OUTPUT            COUNT(\*)  -------------------- ----------  11-NOV-2015 17:57        100000  Elapsed: 00:00:00.21  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SIMILAR TIME!!!  udf does not help    so back to my original function I had issues with  a difference with the multiple vcs in func and my orig func is my orig had defaults  thus I will try a version with no defaults  TEXT\_OUTPUT                      COUNT(\*)  ------------------------------ ----------  Ms Wyyyyyyyyy W Wyyyyyyyyy         100000  Elapsed: 00:00:00.19    TEXT\_OUTPUT                      COUNT(\*)  ------------------------------ ----------  Ms Wyyyyyyyyy W Wyyyyyyyyy         100000  Elapsed: 00:00:00.08  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*UDF has an IMPACT    now with one of the parameters set to a default  TEXT\_OUTPUT                      COUNT(\*)  ------------------------------ ----------  Ms Wyyyyyyyyy W Wyyyyyyyyy         100000  Elapsed: 00:00:00.32    TEXT\_OUTPUT                      COUNT(\*)  ------------------------------ ----------  Ms Wyyyyyyyyy W Wyyyyyyyyy         100000  Elapsed: 00:00:00.32  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*UDF has NO IMPACT  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ALSO defaults cause both versions to be slower    now call the simple disp\_name\_udf function that benefits standalone from within a package    standalone  TEXT\_OUTPUT                      COUNT(\*)  ------------------------------ ----------  Ms Wyyyyyyyyy W Wyyyyyyyyy         100000  Elapsed: 00:00:00.08    within package  TEXT\_OUTPUT                      COUNT(\*)  ------------------------------ ----------  Ms Wyyyyyyyyy W Wyyyyyyyyy         100000  Elapsed: 00:00:00.08  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* WORKS - so long as neither spec of body have prm defaults    and just to round of, using a subquery factored function which my prior tests showed reduced overhead  TEXT\_OUTPUT                      COUNT(\*)  ------------------------------ ----------  Ms Wyyyyyyyyy W Wyyyyyyyyy         100000  Elapsed: 00:00:00.10    the WITH function benefits even with defaults  TEXT\_OUTPUT                      COUNT(\*)  ------------------------------ ----------  Ms Wyyyyyyyyy W Wyyyyyyyyy         100000  Elapsed: 00:00:00.10 |

Creating the test table

|  |
| --- |
| drop table pers;  create table pers  (pers\_id             number(8)    not null  ,surname             varchar2(30) not null  ,first\_forename      varchar2(30) not null  ,second\_forename     varchar2(30)  ,pers\_title          varchar2(10)  ,sex\_ind             char(1)      not null  ,dob                 date  ,addr\_id             number(8)  ,pers\_comment        varchar2(2000)  )  /  insert into pers  select rownum        ,rpad(chr(65+mod(rownum,24)),10,chr(65+mod(rownum,25)))        ,rpad(chr(65+mod(rownum,24)),10,chr(65+mod(rownum,25)))        ,rpad(chr(65+mod(rownum,24)),10,chr(65+mod(rownum,25)))        ,decode(mod(rownum,4),0,'MR',1,'MRS',2,'Ms',3,'MR','DR')        ,decode(mod(rownum,2),0,'M',1,'F')        ,sysdate - (3000+mod(rownum,30000))        ,rownum +1001        ,rpad(chr(65+mod(rownum,24)),200,chr(65+mod(rownum,25)))  from dual  connect by level < 100001  / |

Creating the functions and a small package

|  |
| --- |
| --num\_num  CREATE OR REPLACE FUNCTION normal\_num\_num(p\_id IN NUMBER) RETURN NUMBER IS  v\_num number;  BEGIN    v\_num:=p\_id\*p\_id;    RETURN v\_num;  END;  /  CREATE OR REPLACE FUNCTION udf\_num\_num(p\_id IN NUMBER) RETURN NUMBER IS  PRAGMA UDF;  v\_num number;  BEGIN    v\_num:=p\_id\*p\_id;    RETURN v\_num;  END;  /  --  -- vc\_vc  CREATE OR REPLACE FUNCTION normal\_vc\_vc(p\_id IN varchar2) RETURN varchar2 IS  v\_vc varchar2(100);  BEGIN    v\_vc:=substr(p\_id,1,5);    RETURN v\_vc;  END;  /  CREATE OR REPLACE FUNCTION udf\_vc\_vc(p\_id IN varchar2) RETURN varchar2 IS  PRAGMA UDF;  v\_vc varchar2(100);  BEGIN    v\_vc:=substr(p\_id,1,5);    RETURN v\_vc;  END;  /  -- vc\_vc\_2  CREATE OR REPLACE FUNCTION normal\_vc\_vc\_2(p\_id1 IN varchar2,p\_id2 IN varchar2) RETURN varchar2 IS  v\_vc varchar2(100);  BEGIN    v\_vc:=substr(p\_id1,1,5)||substr(p\_id2,2,5);    RETURN v\_vc;  END;  /  CREATE OR REPLACE FUNCTION udf\_vc\_vc\_2(p\_id1 IN varchar2,p\_id2 IN varchar2) RETURN varchar2 IS  PRAGMA UDF;  v\_vc varchar2(100);  BEGIN    v\_vc:=substr(p\_id1,1,5)||substr(p\_id2,2,5);    RETURN v\_vc;  END;  /  --  -- dt\_dt  CREATE OR REPLACE FUNCTION normal\_dt\_dt(p\_id IN date) RETURN date IS  v\_dt date;  BEGIN    v\_dt:=p\_id+1000;    RETURN v\_dt;  END;  /  CREATE OR REPLACE FUNCTION udf\_dt\_dt(p\_id IN date) RETURN date IS  PRAGMA UDF;  v\_dt date;  BEGIN    v\_dt:=p\_id-1000;    RETURN v\_dt;  END;  /  -- dt\_num  CREATE OR REPLACE FUNCTION normal\_dt\_num(p\_id IN date) RETURN number IS  v\_num number;  BEGIN    v\_num:=to\_char(p\_id,'J')+100;    RETURN v\_num;  END;  /  CREATE OR REPLACE FUNCTION udf\_dt\_num(p\_id IN date) RETURN number IS  PRAGMA UDF;  v\_num number;  BEGIN    v\_num:=to\_char(p\_id,'J')-100;    RETURN v\_num;  END;  /  -- num\_dt  CREATE OR REPLACE FUNCTION normal\_num\_dt(p\_id IN number) RETURN DATE IS  v\_dt date;  BEGIN    v\_dt:=sysdate+(p\_id/100);    RETURN v\_dt;  END;  /  CREATE OR REPLACE FUNCTION udf\_num\_dt(p\_id IN number) RETURN DATE IS  PRAGMA UDF;  v\_dt date;  BEGIN    v\_dt:=sysdate-(p\_id/100);    RETURN v\_dt;  END;  /  create or replace function normal\_disp\_name (p\_sn      in varchar2                         ,p\_fn1     in varchar2                         ,p\_fn2     in varchar2                         ,p\_title   in varchar2  ) return varchar2 is  v\_return     varchar2(1000);  begin    v\_return := case when p\_title is null then ''                     else initcap(p\_title)||' '                end              ||initcap(p\_fn1)||' '              ||case when p\_fn2 is null then ''                     else substr(p\_fn2,1,1)||' '                end              ||initcap(p\_sn);  return v\_return;  end;  /  create or replace function udf\_disp\_name (p\_sn      in varchar2                         ,p\_fn1     in varchar2                         ,p\_fn2     in varchar2                         ,p\_title   in varchar2  ) return varchar2 is  PRAGMA UDF;  v\_return     varchar2(1000);  begin    v\_return := case when p\_title is null then ''                     else initcap(p\_title)||' '                end              ||initcap(p\_fn1)||' '              ||case when p\_fn2 is null then ''                     else substr(p\_fn2,1,1)||' '                end              ||initcap(p\_sn);  return v\_return;  end;  /  create or replace function normal\_disp\_name\_defaults (p\_sn      in varchar2                         ,p\_fn1     in varchar2                         ,p\_fn2     in varchar2                         ,p\_title   in varchar2 :=null  ) return varchar2 is  v\_return     varchar2(1000);  begin    v\_return := case when p\_title is null then ''                     else initcap(p\_title)||' '                end              ||initcap(p\_fn1)||' '              ||case when p\_fn2 is null then ''                     else substr(p\_fn2,1,1)||' '                end              ||initcap(p\_sn);  return v\_return;  end;  /  create or replace function udf\_disp\_name\_defaults (p\_sn      in varchar2                         ,p\_fn1     in varchar2                         ,p\_fn2     in varchar2                         ,p\_title   in varchar2 :=null ) return varchar2 is  PRAGMA UDF;  v\_return     varchar2(1000);  begin    v\_return := case when p\_title is null then ''                     else initcap(p\_title)||' '                end              ||initcap(p\_fn1)||' '              ||case when p\_fn2 is null then ''                     else substr(p\_fn2,1,1)||' '                end              ||initcap(p\_sn);  return v\_return;  end;  /  create or replace package t\_pkg as  function udf\_disp\_name (p\_sn      in varchar2                         ,p\_fn1     in varchar2                         ,p\_fn2     in varchar2                         ,p\_title   in varchar2 ) return varchar2;  end t\_pkg;  /  create or replace package body t\_pkg as  function udf\_disp\_name (p\_sn      in varchar2                         ,p\_fn1     in varchar2                         ,p\_fn2     in varchar2                         ,p\_title   in varchar2 ) return varchar2 is  PRAGMA UDF;  v\_return     varchar2(1000);  begin    v\_return := case when p\_title is null then ''                     else initcap(p\_title)||' '                end              ||initcap(p\_fn1)||' '              ||case when p\_fn2 is null then ''                     else substr(p\_fn2,1,1)||' '                end              ||initcap(p\_sn);  return v\_return;  end;  end t\_pkg;  / |

The test script

|  |
| --- |
| -- udf\_tests  set lines 100 pages 50  set feed off  col text\_output     form a30  col number\_output   form 99999999999  col date\_output     form a20  spool udf\_tests.lst  prompt  running udf\_tests  prompt  set pause off  set autotrace off  set timi on  prompt investigating why pragma udf helps some simple functions and not others  prompt -----------------------------------------------------------------------  --  prompt  --  prompt simple number in-number out function  select /\* mdw\_16a \*/       max(normal\_num\_num(pers\_id)) number\_output         ,count(\*) from pers  /  select /\* mdw\_16b \*/        max(udf\_num\_num(pers\_id)) number\_output         ,count(\*)  from pers  /  prompt  prompt \*\* udf helps  prompt  --  prompt simple varchar in-varchar out function  select /\* mdw\_16c \*/       max(normal\_vc\_vc(surname)) text\_output         ,count(\*) from pers  /  select /\* mdw\_16d \*/       max(udf\_vc\_vc(surname)) text\_output         ,count(\*)  from pers  /  prompt  prompt \*\* udf helps  prompt  --  --  prompt  two varchar in-varchar out function, is the issue with more than one in parameter?  select /\* mdw\_16e \*/       max(normal\_vc\_vc\_2(surname,first\_forename)) text\_output         ,count(\*) from pers  /  select /\* mdw\_16f \*/       max(udf\_vc\_vc\_2(surname,first\_forename)) text\_output         ,count(\*)  from pers  /  prompt  prompt \*\* udf helps  prompt  --  prompt simple date in-date out function  select /\* mdw\_16g \*/       max(normal\_dt\_dt(DOB)) date\_output         ,count(\*) from pers  /  select /\* mdw\_16h \*/       max(udf\_dt\_dt(DOB)) date\_output         ,count(\*)  from pers  /  prompt \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SIMILAR TIME!!!  prompt udf does not help  prompt    --  prompt is date out the issue,  date in-num out function  select /\* mdw\_16i \*/       max(normal\_dt\_num(DOB)) number\_output         ,count(\*) from pers  /  select /\* mdw\_16j \*/       max(udf\_dt\_num(DOB)) number\_output         ,count(\*)  from pers  /  prompt \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SIMILAR TIME!!!  prompt udf does not help    --  prompt is date in the issue,  num in-date out function  select /\* mdw\_16k \*/       max(normal\_num\_dt(pers\_id)) date\_output         ,count(\*) from pers  /  select /\* mdw\_16l \*/       max(udf\_num\_dt(pers\_id)) date\_output         ,count(\*)  from pers  /  prompt \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SIMILAR TIME!!!  prompt udf does not help  --  --  prompt  prompt so back to my original function I had issues with  prompt a difference with the multiple vcs in func and my orig func is my orig had defaults  prompt thus I will try a version with no defaults  prompt  select /\* mdw\_16m \*/          max(normal\_disp\_name(surname,first\_forename,second\_forename,pers\_title)) text\_output         ,count(\*) from pers  /  select /\* mdw\_16n \*/          max(udf\_disp\_name(surname,first\_forename,second\_forename,pers\_title)) text\_output         ,count(\*)  from pers  /  prompt \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*UDF has an IMPACT  prompt  prompt  prompt now with one of the parameters set to a default  select /\* mdw\_16o \*/          max(normal\_disp\_name\_defaults(surname,first\_forename,second\_forename,pers\_title)) text\_output         ,count(\*) from pers  /  select /\* mdw\_16p \*/          max(udf\_disp\_name\_defaults(surname,first\_forename,second\_forename,pers\_title)) text\_output         ,count(\*)  from pers  /  prompt \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*UDF has NO IMPACT  prompt \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ALSO defaults cause both versions to be slower  prompt  prompt now call the simple disp\_name\_udf function that benefits standalone from within a package  prompt  prompt standalone  select /\* mdw\_16q \*/          max(udf\_disp\_name(surname,first\_forename,second\_forename,pers\_title)) text\_output         ,count(\*)  from pers  /  prompt  prompt within package  select /\* mdw\_16r \*/          max(t\_pkg.udf\_disp\_name(surname,first\_forename,second\_forename,pers\_title)) text\_output         ,count(\*)  from pers  /  prompt \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* WORKS - so long as neither spec of body have prm defaults  prompt  prompt and just to round of, using a subquery factored function which my prior tests showed reduced overhead  with    function l\_disp\_name(p\_sn      in varchar2                        ,p\_fn1     in varchar2                        ,p\_fn2     in varchar2                        ,p\_title   in varchar2 )  return varchar2  is  v\_return     varchar2(1000);  begin    v\_return := case when p\_title is null then ''                     else initcap(p\_title)||' '                end              ||initcap(p\_fn1)||' '              ||case when p\_fn2 is null then ''                     else substr(p\_fn2,1,1)||' '                end              ||initcap(p\_sn);  return v\_return;  end l\_disp\_name;  select /\*mdw\_16s \*/          max(l\_disp\_name(p\_sn =>surname           ,p\_fn1  =>first\_forename                         ,p\_fn2=>second\_forename   ,p\_title=>pers\_title)        ) text\_output         ,count(\*)  from pers  /    prompt the WITH function benefits even with defaults  with    function l\_disp\_name(p\_sn      in varchar2                        ,p\_fn1     in varchar2                        ,p\_fn2     in varchar2 :=null                        ,p\_title   in varchar2 :=null )  return varchar2  is  v\_return     varchar2(1000);  begin    v\_return := case when p\_title is null then ''                     else initcap(p\_title)||' '                end              ||initcap(p\_fn1)||' '              ||case when p\_fn2 is null then ''                     else substr(p\_fn2,1,1)||' '                end              ||initcap(p\_sn);  return v\_return;  end l\_disp\_name;  select /\*mdw\_16t \*/          max(l\_disp\_name(p\_sn =>surname           ,p\_fn1  =>first\_forename                         ,p\_fn2=>second\_forename   ,p\_title=>pers\_title)        ) text\_output         ,count(\*)  from pers  /  --  spool off |