Performance Tesing PL/SQL – D R A F T -

Author: Marcel Kratochvil
Date: March 2006

Abstract
Assumptions about PL/SQL and how to program efficiently in it have been quoted, stated and talked about for a long time. With Oracle 10gR2, it is time to relook at core coding techniques and answer once and for all what is efficient, what works and what techniques one might need to consider when programming. The results are suprising. Nothing is taken for granted with fundamental assumptions questioned.

Some of the questions asked include :

- Is PLS_INTEGER better than INTEGER?
- Are tables with a large number of columns in them efficient?
- Is CASE faster or slower than IF-THEN?
- How fast can the database read in a text file?
- Is IF-THEN better than NVL?
- How quick is string comparison?
- Is using Global variables better than parameter passing?
- Do exception handlers impact performance?

This presentation will cover 15 very important PL/SQL coding techniques and look at how fast it is to program using them. This review is done on both Windows and Linux with the goal to work out efficient and scalable programming techniques. To reassess previous assumptions made about PL/SQL and to determine what works best when programming in PL/SQL on Oracle 10gR2.

The environment

Oracle 10.2.0.1 running on Linux with 2Gb of memory. Intel P4, 32 bit.
Oracle 10.2.0.1 running on a Dell Laptop, Pentium M chip with 1.5Gb Memory, 32bit on Windows XP.

The native compiler was not used, just genuine PL/SQL compilation as built into the Oracle database.

Some thoughts on testing

The testing that was done wasn't done to show how fast PL/SQL is, or whether Windows is better than Linux (or vica versa). It was done to determine what are the best techniques for developing Mod PL/SQL Web based applications.

Since working with Oracle5 and also working with PL/SQL since Oracle7, certain habits, techniques and assumptions have been developed and then evolved over time. A number of these came about due to bugs and behaviour of PL/SQL in older releases. With the advent of a newer, faster compiler, it was time to test out these assumptions and answer once and for all what works best.

Conclusions are drawn at the end of each test, but it is important to note that sometimes programming using the most efficient technique isn't always the best way to program. It

depends on how often the code is run and how long it takes to run. For example, it is possible to spend a lot of time tuning a PL/SQL program to run in 9 seconds, when previously it ran in 10 seconds. The tuning is done at the expense of program maintainability. If that program is only run once a day then the efficiency gained is not there.

When looking at the program written there might be confusion in why I have running a number of "junk" statements. The programs when written were designed to bypass any built in optimisation that might be in the compiler, just to ensure what was being tested, was being tested. Dummy statements that randomised actions were put in to ensure the PL/SQL run time engine actually did some work and didn't bypass any clauses which might have skewed the results.

The Testing Environment

For each testing case a program was written. Its not elegent and nicely written, but it does the job, and speaking with a DBA hat on, well - that's what counts. You will not find any comments in it or foreign keys (again I slander this sacred cow without any explanation). If you need comments to help you out to understanding what is happening, then sorry, but this paper isn't for you. Nasty yes, but I need to stress that I am first and foremost a DBA and not a Developer. And every good programmer knows that DBAs are nasty and mean individuals who have a donut fixation. Which funnily enough, describes me accurately.

The appendix at the back of this document lists all the programs used. You can use them, play with them and enhance then extend them. Please try the tests and if you can prove that my results are not consistent or my conclusions wrongly drawn then let me know. Of course on your machine you will not get the same timings. These results are valid as of 10.2.0.1 and could change with any patch release.

A shell was also built to call each program and store information about the running time of the programs.

A warning. Do not draw the conclusion from these results that Linux is faster than Windows or Windows is faster than Linux, and use this as a way of stating Unix is the best or Windows is wonderful and we should all use it. That's a dangerous conclusion to draw, especially as any good DBA/Developer knows that VMS is way better than Windows and Unix combined (it just didn't run as fast as it could have and I live in my little fantasy world). I have no love for Windows or Unix, I tolerate them both equally as one should, without getting attached to them.

When looking at the results, realise that whether it is Windows or Linux, PL/SQL runs really fast, probably faster than you ever expected. It would be nice to try and compare it to a VB program (.asp), Java, Perl or other program and see how fast it really is, or whether I am living in a PL/SQL biaised fantasy world.

All tests were run a number of times to reduce the impact of caching and negate the effect of my MP3 player running on the Windows laptop whilst doing these tests.

What is the question?

In doing these tests its easy to get an answer, but then one should review the answer compared to the initial question and ask, what is the actual question? Its possible the program that ran isn't addressing the question being asked. Its easy for this to happen. The wrong conclusion can then be drawn.

I try to ensure that my conclusions are modest in what they say even though I have not put them through a legal team so they can be embedded in disclaimers. I am aware that it is possible people will use these results to justify their own conclusions whilst embroiled in some in-house political situation, which is usually one where the DBAs and Developers are at each others throats. If this is the case then please use these results to justify the conclusion you desire (its called reverse psychology).

Test 1: Which is faster, CASE or IF-THEN-ELSE ?

Some habits are hard to break, and for me, not using a CASE statement is one of them. CASE only came in a later release of Oracle8. Making the switch to using them was as hard as someone switching from Diet Coke to Coke Zero.

But the statement is there, it is easy to use and very powerful, so what was holding me back from using it? Inherent laziness was the first thing, but beyond that was the nagging suspicion that using it might slow things down. So what better statement to test first than answering that important question, is CASE fast, slow or no difference?

On thinking about it, the compiler should convert a CASE statement into an IF-THEN statement, so common thinking says that they should perform the same. But if only that was the case.

Results:

	Iterations	***Windows*** *(seconds)*	***Linux*** *(seconds)*
Case	1 million	2.67	2.09
If-Then	1 million	2.64	1.94
Case	10 million	25.9	21.0
If-Then	10 million	24.8	19.4

Conclusion:

Well the results are suprising, consistently the CASE statement is slower. Not much slower, but enough that if you were doing a major warehouse load of data on a regular basis, loading millions of record, I would use IF-THEN instead of case. If I had a PL/SQL program that was being run hundreds of thousands of times a day then I would use IF-THEN.

But for general day to day programs, CASE is OK to use. So don't use this as an excuse not to switch to CASE, make the switch and make your code easier to maintain, but for performance, serious performance I would use IF-THEN.
And did you notice how fast it ran? 10 million iterations and checks in under 24 seconds. That's pretty fast on both Windows and Linux. In fact most of the tests done regularly do complex calculations going beyond 1 million iterations.

Test 2: Recursion vs Iteration – which is fastest?

The assumption I was always under was do not use recursion as its very slow, and an iterative loop is always faster. Recursion lends itself to elegent programming and since the sixties with the use of Prolog and Lisp there has always been a wonderful mathematical programming style that was attractive to programming and using recursion.

But for recursion to happen, for every recursive call, stack information needs to be stored, scoping variable setup and there is a natural performance hit.

The question then is, is that performance hit serious enough to rule out once and for all recursion, or is it safe to use it in general programming?

Results:

	Iterations	***Windows*** *(seconds)*	***Linux*** *(seconds)*
Recursion	1 million	2.74	2.09
Iteration	1 million	2.65	1.93
Recursion	10 million	26.01	20.98
Iteration	10 million	25.06	19.38

Conclusion:

Recursion is as expected slower than iteration, but not much slower. In fact the difference is minor. The iteration algorithm used involved very simple parameter passing, and maybe a future test will look at how efficient the recursive algorithm is when more complex and larger sized parameters are passed.

But do not rule out using recursion because you think it is slower.

Test 3: How fast is PL/SQL in processing a text file ?

Background.

These tests were run multiple times to factor in disk and o/s caching.
NTFS compression was not used on the files.

Results:

Width x Height	***File Size***	***Windows*** *(seconds)*	***Linux*** *(seconds)*
3000 x 13,749	40mb	1.1	1.1
3000 x 27,499	80mb	2.1	5.2
3000 x 54,999	161mb	4.0	10.51
3000 x 110,000	322mb	24.1 / 8 **	23.4
10 x 1,249,999	14.6mb	29.2	19.1
10 x 2,499,999	29.2mb	59.1	38.3
10 x 4,999,999	58.5mb	123.7	77.3
10 x 10,000,000	117.1mb	249.6	154.6

** The effect of caching on Windows is apparent. The first run took 24 seconds, the second run took just 8. There was no equivalent on Linux.

Conclusion:

Processing a text file in PL/SQL is very fast. The width and height of the file does impact the performance. As expected, the more successive I/Os that had to be done the longer the time to process the file.

File processing is determined by disk speed and whether the operating system has cached the file in memory (which Windows does do)

Test 4: Global Parameters vs Passing values via Parameters

Good programming habits which I was taught a long time ago, stressed two things:

1. Never use goto's
2. Never use global variables

Now I have been able to survive without goto statements, and rarely use them. When I do, its during program maintenance. Goto's though are not that bad (unlike foreign keys which are nasty little beasts), and smart use of goto's can replace the need for horribly hard to maintain nested ifs (and off course CASE statements).

As for global variables I have been programming for a long time without them, mainly thinking that they are not needed for performance. In the last two years I have changed my view as I try to build more complex application and maintain code. Global variables are very powerful and useful, but do they slow down an application or speed it up?

Results:

	Iterations	***Windows*** *(seconds)*	***Linux*** *(seconds)*
Local(in)	1,000,000	14.7	13.5
Global	1,000,000	12.2	11.5
Local (in)	10,000,000	145.3	135.8
Global	10,000,000	115.5	119.2
Local (in/out)	10,000,000	145.9	140.6

Conclusion:

I ran the test this time without any idea what the result was. Running it on multiple platforms again and again gave me the same respective result. Global variables (actually the test only looked at Global varchars and not complex data types) are faster than passing them down as parameters. It was a noticeable improvement. Changing the definition from in to in/out made no difference in performance.

From my programming point of view I am not going to abandon the use of parameter passing, but I will be using globals in key areas to improve performance.

A future test will use complex types and see what the performance difference is there.

Test 5: How efficient is a table with 1000 columns in it?

This testing was performed to answer the question, is it more efficient to have a table with 1000 columns in it of char(1) with each column have a T or F value, or is it more efficient to have a varchar2(1000) column with each position indicating its value.

The background to this occurred when it was noticed that when building a web site that is data driven, a lot of conditional capabilities are stored. For example, are style sheets used? are logos displayed? Is the interface no frame? Initially columns representing each condition were used, then for a possible performance enhancement they were grouped into one large varchar column.

Its a hard capability to test, but the testing involves creating a table with 1000 columns and populating it with a serious amount of data. An equivalent table with just a primary key column and the one varchar column is created. Both tables are then cleaned up to ensure there is no row migration or possible caching issues.

The test case involves reading then updating a couple of values in the table. And then doing this a large number of times. Rows are semi-randomly updated. The same equivalent column is updated in both test cases.

Results: With Select and Update

Table	***Iterations***	***Windows*** *(seconds)*	***Linux*** *(seconds)*
1000 columns	1,000,000	116	222
1 varchar	1,000,000	312	558

Results: With Select Only

Table	***Iterations***	***Windows*** *(seconds)*	***Linux*** *(seconds)*
1000 columns	1,000,000	48	67
1 varchar	1,000,000	50	70
1000 columns	10,000,000	480	676
1 varchar	10,000,000	495	704

Conclusion:

The 1 varchar table is more efficient at storing the data. More rows per block, lends itself to more efficient caching and faster performance. The test results though did not back this conclusion up. Having 1000 columns is as efficient as having one varchar2 column.

The problem with the 1 varchar table is that when updating the whole column is updated, all 1000 characters. This is slower, consumes more undo and will ultimately consume more redo (and logging).

The 1000 column table requires more overhead to store the data. Its easier to retrieve a value as only the name is needed.

So the recommendation is if you have an application where you need to store a large set of business logic in the data, its more efficient or as efficient to store one column per value than trying to think you are efficient by storing multiple values in the one column.

Test 6: Time to Encrypt Data

With the move to web enabling applications and storing data on databases that could be vulnerable to hacking, the temptation is to store sensitive data in an encrypted format. In addition, by encrypting the URL it becomes a lot harder for a hacker to perform URL code injection.

So how efficient is data encryption?

This test takes strings of different lengths of random data, then encrypts them and calculates the average time it takes to encrypt.

The test is run a couple of times to negate the effect of initially starting Java inside the database.

Results:

Length	***Iterations***	***Windows*** *(seconds)*	***Linux*** *(seconds)*
8	100,000	1.8	2.5
16	100,000	1.7	2.5
32	100,000	1.9	2.6
64	100,000	2.1	2.9
128	100,000	2.5	3.3
256	100,000	3.4	4.0
512	100,000	5.2	5.4
1024	100,000	8.8	8.3
2048	100,000	15.7	14.0
4096	100,000	30.3	25.4
8192	100,000	57.8	48.6
16384	100,000	112.4	95.0
32760	100,000	224.6	188.8

Conclusion:

Encryption is very fast. To protect a web based application from URL code injection, it is recommended that key values are always encrypted.

Test 7: Does nvl perform at the same speed as if then ?

This was a trivia test. After doing a number of tests the idea came to look at nvl and see how well it performs. I was expecting it to perform the same as if then, as the optimiser should be able to break the nvl down to an if then on compilation. So was that assumption valid?

Results:

	Iterations	***Windows*** *(seconds)*	***Linux*** *(seconds)*
nvl	10,000,000	9.57	9.85
if then	10,000,000	8.29	8.68
nvl	100,000,000	93.3	98.9
if then	100,000,000	80.1	86.9

Conclusion:

The results are conclusive and suprising. The nvl function is slower than an if then statement. Is it slow enough to justify not using it? I would say no, as the performance difference is only noticeable as the number of iterations exceeds an incredibly large and unrealistic number.

Test 8: Is (X and Y) the same in performance as (Y and X) ?

One of the key advances in the Oracle optimser for SQL was the ability for it to analyse a SQL statement and determine in a AND situation which clause should be run first. This eliminated the need to hand tune SQL. Does this same optimisation technique apply to PL/SQL? The test is simple. A simple clause is ANDed with a complex clause designed to run slowly. The simple clause always fails, and if it is run first the whole statement should not run, resulting in fast performance.

To ensure there is no obvious hand optimisation, the simple statement is non trivial.

Results:

	Iterations	***Windows*** *(seconds)*	***Linux*** *(seconds)*
simple AND complex	1000	0	0
complex AND simple	1000	1.69	1.62
simple OR complex	1000	0	0
complex OR simple	1000	1.62	1.62

Conclusion:

There is no optimisation with AND or OR statements. The statements are run left to right. As such, it is important to put the statement that runs the fastest and likely to return a fail (if AND) or succeed (if OR) as the left clause.

Test 9: Efficiency of Numbers

This test looks at the performance improvements made when using the different number types available, in particular a review is made of the new *binary_float* and *binary_double* values that come with 10gR2.

Results:

Variable	***Iterations***	***Windows*** *(seconds)*	***Linux*** *(seconds)*
integer	100,000	7.98	10.25
pls_integer	100,000	8.03	10.19
binary_integer	100,000	7.99	10.20
binary_float	100,000	0.31	0.28
binary_double	100,000	0.36	0.25
number	100,000	8.59	10.90

Conclusion:

The binary_float and binary_double are so much faster doing calculations in comparison to the other variables, that it is not even worthwhile talking about whether integer is better or worse than pls_integer.

It is recommended that any variable involving simple to complex calculations be declared as a binary_float or binary_double. This should be reviewed when migrating an Oracle database to 10gR2. Side effects of using these new variables is yet to be seen or determined.

Test 10: String Comparison – Failed Match

When comparing a small string to a large one, and the test will fail the comparison, does the length of the string matter? The assumption is that it shouldn't.

Results:

String Length	***Iterations***	***Windows*** *(seconds)*	***Linux*** *(seconds)*
3	10,000,000	6.3	6.1
10	10,000,000	6.2	6.1
100	10,000,000	6.2	6.1
1000	10,000,000	6.2	6.1
10,000	10,000,000	6.2	6.1

Conclusion:

What was expected is what happened. The size of the string did not impact the performance. Finally an assumption wasn't proved wrong.

Test 11: String Comparison – exact match

This test looks at the efficiency of PL/SQL when it comes to performing string comparison tests, as the size of the string increases in size. This test continues from test 10, but in this case testing for an exact match is more expensive than a failed match. The assumption is that as the string size increases so does the time to do the calculation.

Results:

String Size	***Iterations***	***Windows*** *(seconds)*	***Linux*** *(seconds)*
3	10,000,000	9.0	8.8
10	10,000,000	9.1	9.2
100	10,000,000	10.1	9.5
1000	10,000,000	17.9	14.3
10,000	10,000,000	116.0	77.3

Conclusion:

String comparisons up to a length of 100 characters seems consistent in performance. Moving up to 1000 characters there is a noticeable performance impact and going to 10,000 characters there is a significant performance drop.

The recommendation is to keep string comparison checks short and under 1000 characters in length. Beyond this length, if possible try doing different comparison checks.

Test 12: Mod PL/SQL – Is htp.p('<TD>..</TD>') faster than htp.tabledata('..') ?

The Mod PL/SQL package has been in use since 1995 and different programming styles have emerged as the package grew. Some programmers insisted on using all the supplied functions, such as htp.bold and htp.frameset, whilst others decided to encode raw html in the htp.p procedure.

For those who are not familiar with the htp package, when looking at the raw code all htp.x procedures eventually boil down to a htp.p call which creates the raw html.

There are advantages and disadvantages to each and they are not going to be covered here. What we are looking at is, which is fastest?

Results:

	Iterations	***Windows*** *(seconds)*	***Linux*** *(seconds)*
htp.p('<TD...	1,000,000	13.1	17.5
htp.tabledata('...	1,000,000	15.1	20.6

Conclusion:

The htp.tabledata command has to do a PL/SQL procedure call that then calls the htp.p procedure.

For performance in a mod PL/SQL application, it is best to use raw htp commands.
This might come at the expense of code management and maintenance.

Test 13: Mod PL/SQL: Clob vs Procedure

This test is complicated in what it is trying to show. When using Mod PL/SQL and the htp package, all information is written to the *owa* array. When the procedure is finished, the array is sent to the browser.

Another method that is not well known, is to use wpg_docload and pass a binary file back to the browser. This file can contain HTML data as well as binary/image data. By populating a Clob with raw HTML an alternate method to the htp package is made available. The question is, is it faster?

Results:

	Iterations	***Windows*** *(seconds)*	***Linux*** *(seconds)*
htp	1,000,000	13.6	18.4
clob	1,000,000	5.5	3.8
htp	5,000,000	52.9	95.3
clob	5,000,000	25.2	42.2

Conclusion:

What happens is the more iterations the more memory is required to store the array used by the htp package. As the clob is stored in memory and disk, this caching is managed by Oracle and is more efficient.

When using htp, if the size of the page exceeds 1mb then maybe it might be best to use a clob to store the page rather than using the htp package.

Test 14: Array Searching vs PL/SQL Temporary table

Background.

The purpose of this test is to try and determine an efficient method for searching for fixed data. The test looks at whether it is quicker to populate an array and search for the data using this method or to populate a temporary table and then perform a SQL query to find it. PL/SQL arrays have a reputation for being slow.

Results:

	File Size	***Windows*** *(seconds)*	***Linux*** *(seconds)*
array	10,000	31.4	41.4
query	10,000	42.3	49.0

Conclusion:

The SQL Query had an unfortunate problem, it always had to do a full scan looking for the results, where as the array stopped processing once it found the first value. This little issue could be enough to tip the balance in favour of the array.

Future tests will try more complex checks and use larger amounts of data. As was shown with clobs and arrays, arrays consume real memory and start to hit realistic memory issues as they get to a certain size, whereas SQL tables do not have this problem. ie. array's do not scale whereas tables do.

Test 15: Is it more efficient to create one large package or have many smaller packages?

When doing web development in PL/SQL it doesn't take long before the packages grow in size and exceed 10,000 lines of code. The question that is asked is, is it better to keep the packages smaller and quicker to compile or is it better for performance to combine all packages into the one super package?

The test involves creating one package with a series of procedures in it. The procedures call each other and do some dummy work. The second test takes these procedures and converts them into packages that then perform the same work.

Results:

	Iterations	***Windows*** *(seconds)*	***Linux*** *(seconds)*
One Super Package	1,000,000	6.2	9.7
Many Smaller Packages	1,000,000	7.0	10.3
One Super Package	10,000,000	62.5	100.4
Many Smaller Packages	10,000,000	71.2	106.3

Conclusion:

Not a huge difference in performance but a noticeable one. For a PL/SQL program that is used frequently and where performance is the goal, and when that program calls other programs, then it is best to move them into one super package to get that performance.

Test 16: Passing clobs as parameters

This test is a review of finding found in Oracle8 concerning innefficiencies found when passing clobs as parameters. What was found is that the more often clobs were passed as parameters the slower the program ran. The interim solution was to use a global clob. This test is a review to see if the problem still exists and what is the most efficient parameter passing method when using clobs.

Results:

	Iterations	***Windows*** *(seconds)*	***Linux*** *(seconds)*
Passing clob down one level using *in out*	100,000	836	575
Passing clob down one level using *in out nocopy*	100,000	4.6	4.5
Passing clob recuresively using *in out ncopy*	100,000	4.8	4.5
Passing clob recurisively using *in out*	10,000	415 seconds and pushed temp tablespace from 500mb to 2.1gb	93 seconds and pushed temp tablespace to 2.1gb
Passing clob using a global variable	100,000	4.6	4.5

Conclusion:

When using clobs and passing them as parameters, it is critical that *in out nocopy* is used instead of just *in out*. The benefit of using a global clob was not seen.

Test 17: Temporary tables, how fast are they?

Global temporary tables have been explained as offering the ability to improve performance because they store their data in a temporary area. This check looks to see how well temporary tables perform.

Results:

	Iterations	***Windows*** *(seconds)*	***Linux*** *(seconds)*
Standard Table	100,000	563	434
Temporary - commit preserve rows	100,000	563	433
Temporary - commit delete rows	1000,000	8	12

Conclusion:

The performance attributed to global temporary tables is only seen when the *commit delete rows* clause in the create table statement is used. This clause prevents the use of undo, as any transaction end command (commit, rollback) always empties the table out.

The recommendation is when using temporary tables is to use the *commit delete rows* clause if you can ensure that the transaction will not do any commit statements within it. This might mean changing the code to achieve this behaviour. The performance benefits seen might outweigh the necessity to change the code.

Test 18: Do exception handlers impact performance?

This test is designed to see if embedding a *begin..exception..end;* within the code will impact the performance of the code. Exception handlers can be quite useful and as they trap any error can save on a lot of error checking and handling.

Results:

	Iterations	***Windows*** *(seconds)*	***Linux*** *(seconds)*
Without exception handler	10,000,000	25.0	30.0
With exception handler	10,000,000	25.0	29.9

Conclusion:

There was no differerence in performance when using an exception handler than not using one. The conclusion is that they can be used nested inside PL/SQL and will not affect performance.

Test 19: Do sub procedures impact performance?

This test looks at whether a procedure within a procedure (nested procedure) is faster or slower compared to pulling the procedure out and putting it at the same level as the parent. Nested procedure have advantages in programming as they can access the parent procedures variables without having to have them passed down as parameters.

Results:

	Iterations	***Windows*** *(seconds)*	***Linux*** *(seconds)*
Nested	10,000,000	44.5	69.6
Not Nested	10,000,000	44.6	68.9

Conclusion:

There was no significant difference between the two test cases. The conclusion is that sub procedures do not improve the performance or degrade it when using them. In which case the requirement for using sub procedures is purely there to improve programming effort by simplifying the code.

Test 20: Does the position of a variable when being declared impact performance?

This is a test I have been always wanting to do just to answer that niggling question which for years has always been bothering me. In web based applications there can be a requirement to declare a lot of variables. Maybe 30 to 50 of them. For aesthetic reasons these variables are grouped together in the declaration section based on type and length. The niggly question is, is this grouping in any way impacting performance? The hope is that it will not, but it is dangerous to take common assumptions for granted.

Results:

	Iterations	***Windows*** *(seconds)*	***Linux*** *(seconds)*
Core variables grouped together	10,000,000	98	63.4
Core variables separated across mixture of complex types	10,000,000	100	65.0
Core variables grouped together	100,000,000	1005	636
Core variables separated across mixture of complex types	100,000,000	1018	650

Conclusion:

Though the difference is not large, there is a small difference in performance, showing that the position of variables in a declaration statement can impact the performance of a PL/SQL application. This test did look at the extremes in variable declaration by creating a large number of large variables, and the difference was noticeable only after 10 million iterations, but the effect is still there and should be considered when scaling applications.

I run this test a number of times just to ensure the effect was actually there and not due to something else, but the results were consistent. Initially the windows results were non conclusive but due to varying nature of the window o/s small background tasks can skew the results, hence the retest on a larger number of iterations. Out of all the tests run so far, this is the only one I am not confident with the answer, and will in future tests run it differently to see whether the same behaviour still appears.

Final Conclusions

What was suprising at the end of the testing was how fast PL/SQL is. When serving web pages from Oracle, PL/SQL is incredibly fast. With the added advantage it runs inside the database, is cached and has built in programming enhancement techniques that make it ideal for web development.

What cannot be explained is the difference in performance between Windows and Linux. Sometimes Windows tests ran faster, sometimes Linux did. As it is the same code base the expectation is that one platform would always run consistently faster than the other, with the winner being driven by the underlying chip speed. This was not seen.

The PL/SQL optimiser does a very good job compiling the code, which is proven by the performance seen, but as a number of tests showed, there are still improvements that can be made, in which case we should be seeing possible improvements in performance with later releases of PL/SQL.

General Conclusions for building web based applications

When it comes to web development, if you have decided that you want to use Oracle as the back end database and you are building a database driven web based application, then the development tools to use must be PL/SQL with additional Java extensions (to do tasks PL/SQL cannot always do).

When thinking of using other tools, such as Perl, Cold Fusion, .ASP, etc the question to be asked is, why do you want to impact the scalability and performance of the application by using these tools?

Appendix – Listing of Programs

```
create or replace procedure a1(run_loop in integer, test_run in char)
as

 vx             varchar2(100);
 vl             varchar2(100);
 va             varchar2(100);

begin

 if test_run = 'X'
  then
    for j in 1..run_loop loop
     vx := to_char(60 + mod(run_loop,100));
     case vx
      when 'A' then vl := 'FFFFFFFFFFFFFFFF';
      when 'B' then vl := 'FFFFFFFFFFFFFFFF';
      when 'C' then vl := 'FFFFFFFFFFFFFFFF';
      when 'D' then vl := 'FFFFFFFFFFFFFFFF';
      when 'E' then vl := 'FFFFFFFFFFFFFFFF';
      when 'F' then vl := 'FFFFFFFFFFFFFFFF';
      when 'G' then vl := 'FFFFFFFFFFFFFFFF';
      when 'H' then vl := 'GGGGGGGGGGGGGGGG';
      when 'I' then vl := 'RRRRRRRRRRRRRRRR';
      when 'J' then vl := 'RRRRRRRRRRRRRRRR';
      when 'K' then vl := 'RRRRRRRRRRRRRRRR';
      when 'L' then vl := 'RRRRRRRRRRRRRRRR';
      when 'M' then vl := 'RRRRRRRRRRRRRRRR';
      when 'N' then vl := 'RRRRRRRRRRRRRRRR';
      when 'O' then vl := 'RRRRRRRRRRRRRRRR';
      when 'P' then vl := 'RRRRRRRRRRRRRRRR';
      when 'Q' then vl := 'RRRRRRRRRRRRRRRR';
      when 'R' then vl := 'RRRRRRRRRRRRRRRR';
      when 'S' then vl := 'RRRRRRRRRRRRRRRR';
      when 'T' then vl := 'RRRRRRRRRRRRRRRR';
      when 'U' then vl := 'RRRRRRRRRRRRRRRR';
      when 'V' then vl := 'RRRRRRRRRRRRRRRR';
      when 'W' then vl := 'RRRRRRRRRRRRRRRR';
      when 'X' then vl := 'RRRRRRRRRRRRRRRR';
      when 'Y' then vl := 'RRRRRRRRRRRRRRRR';
      when 'Z' then vl := 'RRRRRRRRRRRRRRRR';
      else vl := 'MMMMMMMMMMMMMMMM';
      end case;

      va :=
      case
       when vl = 'FFFFFFFFFFFFFFFF' then 'A'
       when vl = 'GGGGGGGGGGGGGGGG' then 'B'
       when vl = 'RRRRRRRRRRRRRRRR' then 'C'
       else 'Z'
      end;

    end loop;
   return;
 end if;

 if test_run = 'Y'
  then
    for j in 1..run_loop loop
     vx := to_char(60 + mod(run_loop,100));
     if vx = 'A' then vl := 'FFFFFFFFFFFFFFFF';
     elsif vx = 'B' then vl := 'FFFFFFFFFFFFFFFF';
     elsif vx = 'C' then vl := 'FFFFFFFFFFFFFFFF';
     elsif vx = 'D' then vl := 'FFFFFFFFFFFFFFFF';
     elsif vx = 'E' then vl := 'FFFFFFFFFFFFFFFF';
     elsif vx = 'F' then vl := 'FFFFFFFFFFFFFFFF';
     elsif vx = 'G' then vl := 'FFFFFFFFFFFFFFFF';
     elsif vx = 'H' then vl := 'GGGGGGGGGGGGGGGG';
     elsif vx = 'I' then vl := 'RRRRRRRRRRRRRRRR';
     elsif vx = 'J' then vl := 'RRRRRRRRRRRRRRRR';
     elsif vx = 'K' then vl := 'RRRRRRRRRRRRRRRR';
     elsif vx = 'L' then vl := 'RRRRRRRRRRRRRRRR';
     elsif vx = 'M' then vl := 'RRRRRRRRRRRRRRRR';
     elsif vx = 'N' then vl := 'RRRRRRRRRRRRRRRR';
     elsif vx = 'O' then vl := 'RRRRRRRRRRRRRRRR';
     elsif vx = 'P' then vl := 'RRRRRRRRRRRRRRRR';
     elsif vx = 'Q' then vl := 'RRRRRRRRRRRRRRRR';
     elsif vx = 'R' then vl := 'RRRRRRRRRRRRRRRR';
     elsif vx = 'S' then vl := 'RRRRRRRRRRRRRRRR';
     elsif vx = 'T' then vl := 'RRRRRRRRRRRRRRRR';
     elsif vx = 'U' then vl := 'RRRRRRRRRRRRRRRR';
     elsif vx = 'V' then vl := 'RRRRRRRRRRRRRRRR';
     elsif vx = 'W' then vl := 'RRRRRRRRRRRRRRRR';
     elsif vx = 'X' then vl := 'RRRRRRRRRRRRRRRR';
     elsif vx = 'Y' then vl := 'RRRRRRRRRRRRRRRR';
     elsif vx = 'Z' then vl := 'RRRRRRRRRRRRRRRR';
      else vl := 'MMMMMMMMMMMMMMMM';
      end if;

     if vl = 'FFFFFFFFFFFFFFFF' then va := 'A';
     elsif vl = 'GGGGGGGGGGGGGGGG' then va := 'B';
     elsif vl = 'RRRRRRRRRRRRRRRR' then va := 'C';
     else va := 'Z';
     end if;

    end loop;

   return;
 end if;

end a1;
/
sho err
```

```
create or replace function a2x(run_loop in integer)
 return varchar2
as

 vx             varchar2(100);
 vl             varchar2(100);
 va             varchar2(100);

begin

 vx := '12345678901234567890';
 if run_loop = 0 then return( vx ); else return( substr(a2x(run_loop - 1) || '0987654321', 1, 90 ) ); end if;

end a2x;
/

create or replace procedure a2(run_loop in integer, test_run in char)
as

 vx             varchar2(100);
 vl             varchar2(100);
 va             varchar2(100);

begin

 if test_run = 'X'
  then
   vx := a2x( run_loop );
   return;
 end if;

 if test_run = 'Y'
  then
   vx := '12345678901234567890';
   for j in 1..run_loop loop
    vx := substr( vx || '0987654321', 1, 90 );
   end loop;
 end if;

end a2;
/

sho err
```

```
create or replace procedure a3(run_loop in integer, test_run in char)
as

 fl             utl_file.file_type;
 bfr            varchar2(32767);
 ctr            integer;
 ctr2           integer;

begin
 ctr := 0;
 ctr2 := 0;

 dbms_output.put_line( 'File:' || 'large_file_' || lower(test_run) || '.txt');
 fl := utl_file.fopen( '/u01/test', 'large_file_' || lower(test_run) || '.txt', 'r', 32767 );
 loop
  begin
   utl_file.get_line(fl,bfr);
   ctr := ctr + 1;
   ctr2 := ctr2 + length(bfr);
  exception
   when others then exit;
  end;
 end loop;
 utl_file.fclose( fl );
 dbms_output.put_line( 'Number of Lines:' || ctr );
 dbms_output.put_line( 'Number of Chars:' || ctr2 );
end a3;
/

sho err
```

```
create or replace procedure a4(run_loop in integer, test_run in char)
as

 MYGLOBAL       varchar2(100);
 mylocal        varchar2(100);

 procedure subproc( alocalparm in out varchar2 )
 as
  x             integer;
  tmp           varchar2(100);
 begin
  -- Just do some dummy work on the local variable
  x := length( alocalparm );
  if instr( alocalparm,'+',5) > 0 then null; end if;
  for k in 1..20 loop
   tmp := substr(alocalparm,k,20);
  end loop;
 end subproc;

 procedure global_subproc
 as
  x             integer;
  tmp           varchar2(100);
 begin
  -- Just do some dummy work on the local variable
  x := length( MYGLOBAL );
  if instr( MYGLOBAL,'+',5) > 0 then null; end if;
  for k in 1..20 loop
   tmp := substr(MYGLOBAL,k,20);
  end loop;
 end global_subproc;

begin

 if test_run = 'X'
  then
    for j in 1..run_loop loop
     mylocal := lpad(to_char(j),100,'+');
     subproc( mylocal );

    end loop;
   return;
 end if;

 if test_run = 'Y'
  then
    for j in 1..run_loop loop
     MYGLOBAL := lpad(to_char(j),100,'+');
     global_subproc;

    end loop;
   return;
 end if;

end a4;
/

sho err
```

```
create or replace procedure a5(run_loop in integer, test_run in char)
as

 cursor c1x( id integer ) is select col_x500 from a5x where pk = id;
 cursor c1y( id integer ) is select substr(col_y,500,1) col_x500 from a5y where pk = id;

 vx             char(1);
 vl             varchar2(100);
 va             varchar2(100);

 -- Redo this test, but for reads and writes separately.
begin

 if test_run = 'X'
  then
    for j in 1..run_loop loop

     -- Retrieve a value
     open c1x( mod(j,10000) );
     fetch c1x into vx;
     close c1x;

     -- Update one column
     /*
     update a5x
      set
       col_x200 = chr(mod(j,60)+60)
     where
      pk = mod(j+987654,10000);
     */
     --
    end loop;
   return;
 end if;

 if test_run = 'Y'
  then
    for j in 1..run_loop loop

     -- Retrieve a value
     open c1y( mod(j,10000) );
     fetch c1y into vx;
     close c1y;

     -- Update one column
     /*
     update a5y
      set
       col_y = substr(col_y,1,199) || chr(mod(j,60)+60) || substr(col_y,201)
     where
      pk = mod(j+987654,10000);
     */
     --
    end loop;

   return;
 end if;

end a5;
/
sho err

create table a5x
(
 pk                                              number(16),
col_x001                        char(1) DEFAULT 'F',
col_x002                        char(1) DEFAULT 'F',
col_x003                        char(1) DEFAULT 'F',
... repeated all the way through
col_x999                        char(1) DEFAULT 'F',
col_x1000                       char(1) DEFAULT 'F');

create table a5y
(
 pk                                              number(16),
 col_y                                           char(1000) DEFAULT lpad('F',1000,'F')
);

begin
 for j in 1..100000 loop
  insert into a5x (pk,col_x001,col_x1000) values (j,'T','T');
 end loop;
end;
/
commit;

begin
 for j in 1..100000 loop
  insert into a5y(pk,col_y) values (j,'T' || lpad('F',998,'F') || 'T');
 end loop;
end;
/
commit;

create index i_a5x_1 on a5x(pk);
create index i_a5y_1 on a5y(pk);
-- This step is done just in case chaining could cause problems.
alter table a5x enable row movement;
alter table a5y enable row movement;

alter table a5x shrink space compact cascade;
alter table a5y shrink space compact cascade;

analyze table a5x compute statistics;
analyze table a5y compute statistics;
```

```
create or replace procedure a6(run_loop in integer, test_run in char)
as

   TYPE intarr IS TABLE OF INTEGER;

   raw_input                   raw(32767);
   raw_key                     raw(8) := utl_raw.cast_to_raw( '12345678' );
   encrypted_raw        raw(32767);
   crec                                    performance_results%ROWTYPE;
   avgt                                    number;
   kloop                       intarr := intarr(8,16,32,64,128,256,512,1024,2048,4096,8192,16384,32760); -- Any value above 32760 and
a ORA-28232: invalid input length for obfuscation toolkit

begin
 crec.test_name := 'A6A';
 crec.invoke_call := null;
 delete from performance_results where test_name = 'A6A';

 for k in kloop.first..kloop.last loop

  crec.run_ctr := kloop(k);
  crec.start_time := systimestamp;

  for j in 1..run_loop loop

   raw_input := utl_raw.cast_to_raw(rpad(to_char(60+j),kloop(k),to_char(70+j)));

   dbms_obfuscation_toolkit.desencrypt(input => raw_input, key => raw_key, encrypted_data => encrypted_raw );

  end loop;

  crec.finish_time := systimestamp;
  avgt := to_number(to_char(crec.finish_time,'SSSSS.FF5')) - to_number(to_char(crec.start_time,'SSSSS.FF5'));
  crec.run_result := to_char(round(avgt,5));
  insert into performance_results values crec;

 end loop;
 commit;
 end a6;
/
sho err
```

```
create or replace procedure a7(run_loop in integer, test_run in char)
as

 vx             varchar2(100);
 vl             varchar2(100);
 va             varchar2(100);

begin

 if test_run = 'X'
  then
    for j in 1..run_loop loop
     vl := nvl(vx,'A');
     if mod(j,2) = 1 then vx := null; else vx := 'X'; end if;
    end loop;
   return;
 end if;

 if test_run = 'Y'
  then
    for j in 1..run_loop loop
     if vx is null then vl := 'A'; else vl := vx; end if;
     if mod(j,2) = 1 then vx := null; else vx := 'X'; end if;
    end loop;

   return;
 end if;

end a7;
/
sho err
```

```
create or replace procedure a8(run_loop in integer, test_run in char)
as

 vx             varchar2(100);
 vl             varchar2(100);
 va             varchar2(100);

 function call_complex( ctr in integer )
  return boolean
 as
  x integer;
 begin
  for k in 1..10000 loop
   x := k;
  end loop;
  return( FALSE );
 end call_complex;

 function call_complex_true( ctr in integer )
  return boolean
 as
  x integer;
 begin
  for k in 1..10000 loop
   x := k;
  end loop;
  return( TRUE );
 end call_complex_true;

begin

 if test_run = 'X'
  then
    for j in 1..run_loop loop
     if (j=0) and ( call_complex(j) )
      then
       vx := 'X';
     end if;
    end loop;
   return;
 end if;

 if test_run = 'Y'
  then
    for j in 1..run_loop loop
     if ( call_complex(j) ) and (j=0)
      then
       vx := 'Y';
     end if;
    end loop;
   return;
 end if;

 if test_run = 'A'
  then
    for j in 1..run_loop loop
     if (j+1=j+1) or ( call_complex(j) )
      then
       vx := 'A';
     end if;
    end loop;
   return;
 end if;

 if test_run = 'B'
  then
    for j in 1..run_loop loop
     if ( call_complex_true(j) ) or (j+1=j+1)
      then
       vx := 'B';
     end if;
    end loop;
   return;
 end if;

end a8;
/
sho err
```

```
create or replace procedure a9(run_loop in integer, test_run in char)
as

 --1
 v_int_a        integer;
 v_int_b        integer;
 v_int_t        integer;

 --2
 v_pls_a        pls_integer;
 v_pls_b        pls_integer;
 v_pls_t        pls_integer;

 --3
 v_bin_a        binary_integer;
 v_bin_b        binary_integer;
 v_bin_t        binary_integer;

 --4
 v_flt_a        binary_float;
 v_flt_b        binary_float;
 v_flt_t        binary_float;

 --5
 v_dbl_a        binary_double;
 v_dbl_b        binary_double;
 v_dbl_t        binary_double;

 --6
 v_nmb_a        number;
 v_nmb_b        number;
 v_nmb_t        number;

begin

 if test_run = '1'
  then
    v_int_t := 0;
    for j in 1..run_loop loop
      v_int_a := j;
      v_int_b := v_int_a * 1234;
      v_int_b := sqrt( v_int_b );
      v_int_b := v_int_b * v_int_b;
      v_int_b := cos( v_int_b );
      v_int_b := tan(v_int_a) * power(v_int_b, j );
      v_int_t := v_int_t + v_int_b;
    end loop;
    dbms_output.put_line( 'Result:' || v_int_t );
   return;
 end if;

 if test_run = '2'
  then
    v_pls_t := 0;
    for j in 1..run_loop loop
      v_pls_a := j;
      v_pls_b := v_pls_a * 1234;
      v_pls_b := sqrt( v_pls_b );
      v_pls_b := v_pls_b * v_pls_b;
      v_pls_b := cos( v_pls_b );
      v_pls_b := tan(v_pls_a) * power(v_pls_b, j );
      v_pls_t := v_pls_t + v_pls_b;
    end loop;
    dbms_output.put_line( 'Result:' || v_pls_t );
   return;
 end if;

 if test_run = '3'
  then
    v_bin_t := 0;
    for j in 1..run_loop loop
      v_bin_a := j;
      v_bin_b := v_bin_a * 1234;
      v_bin_b := sqrt( v_bin_b );
      v_bin_b := v_bin_b * v_bin_b;
      v_bin_b := cos( v_bin_b );
      v_bin_b := tan(v_bin_a) * power(v_bin_b, j );
      v_bin_t := v_bin_t + v_bin_b;
    end loop;
    dbms_output.put_line( 'Result:' || v_bin_t );
   return;
 end if;

 if test_run = '4'
  then
    v_flt_t := 0;
    for j in 1..run_loop loop
      v_flt_a := j;
      v_flt_b := v_flt_a * 1234;
      v_flt_b := sqrt( v_flt_b );
      v_flt_b := v_flt_b * v_flt_b;
      v_flt_b := cos( v_flt_b );
      v_flt_b := tan(v_flt_a) * power(v_flt_b, j );
      v_flt_t := v_flt_t + v_flt_b;
    end loop;
    dbms_output.put_line( 'Result:' || v_flt_t );
   return;
 end if;

 if test_run = '5'
  then
    v_dbl_t := 0;
    for j in 1..run_loop loop
      v_dbl_a := j;
      v_dbl_b := v_dbl_a * 1234;
      v_dbl_b := sqrt( v_dbl_b );
      v_dbl_b := v_dbl_b * v_dbl_b;
```

```
      v_dbl_b := cos( v_dbl_b );
      v_dbl_b := tan(v_dbl_a) * power(v_dbl_b, j );
      v_dbl_t := v_dbl_t + v_dbl_b;
    end loop;
    dbms_output.put_line( 'Result:' || v_dbl_t );
   return;
 end if;

 if test_run = '6'
  then
    v_nmb_t := 0;
    for j in 1..run_loop loop
      v_nmb_a := j;
      v_nmb_b := v_nmb_a * 1234;
      v_nmb_b := sqrt( v_nmb_b );
      v_nmb_b := v_nmb_b * v_nmb_b;
      v_nmb_b := cos( v_nmb_b );
      v_nmb_b := tan(v_nmb_a) * power(v_nmb_b, j );
      v_nmb_t := v_nmb_t + v_nmb_b;
    end loop;
    dbms_output.put_line( 'Result:' || v_nmb_t );
   return;
 end if;


end a9;
/
sho err
```

```
create or replace procedure a10(run_loop in integer, test_run in char)
as

 vx             varchar2(100);
 vl             varchar2(100);
 va             varchar2(100);

begin

 if test_run = 'A'
  then
    for j in 1..run_loop loop
     vx := 'X' || to_char(j) || 'Y';
     if vx = 'A'
      then
       vl := '1';
      else
       vl := '2';
     end if;
    end loop;
   return;
 end if;

 if test_run = 'B'
  then
    for j in 1..run_loop loop
     vx := 'X' || to_char(j) || 'Y';
     if vx = 'A234567890'
      then
       vl := '1';
      else
       vl := '2';
     end if;
    end loop;
   return;
 end if;

 if test_run = 'C'
  then
    for j in 1..run_loop loop
     vx := 'X' || to_char(j) || 'Y';
     if vx = 'A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890'
      then
       vl := '1';
      else
       vl := '2';
     end if;
    end loop;
   return;
 end if;

 if test_run = 'D'
  then
    for j in 1..run_loop loop
     vx := 'X' || to_char(j) || 'Y';
     if vx =
'A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2
34567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345
67890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345678
90A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A
234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234
567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567
890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890
A234567890A234567890A234567890A234567890A234567890A234567890A234567890'
      then
       vl := '1';
      else
       vl := '2';
     end if;
    end loop;
   return;
 end if;

 if test_run = 'E'
  then
    for j in 1..run_loop loop
     vx := 'X' || to_char(j) || 'Y';
     if vx =
'A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2
34567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345
67890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345678
90A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A
234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234
567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567
890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890
A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23
4567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23456
7890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23456789
0A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2
34567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345
67890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345678
90A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A
234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345
     67890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23
4567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23456
7890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23456789
0A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2
34567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345
67890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345678
90A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A
234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234
567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567
890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890
A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23
4567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23456
7890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23456789
0A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2
```

```
34567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345
6789
     0A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567
890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890
A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23
4567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23456
7890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23456789
0A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2
34567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345
67890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345678
90A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A
234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234
567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567
890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890
A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23
4567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23456
7890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345678
90A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A
234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234
567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567
890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890
A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23
4567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23456
7890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345678
90A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A
234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234
567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567
890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890
A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23
4567
     890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345
67890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A
234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234
567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567
890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890
A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23
4567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23456
7890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345678
90A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A
234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234
567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567
890A
     234567890A234567890A234567890'
      then
       vl := '1';
      else
       vl := '2';
     end if;
    end loop;
   return;
 end if;

end a10;
/
sho err
```

```
create or replace procedure a11(run_loop in integer, test_run in char)
as

 vl             varchar2(100);
 va             varchar2(32767);
 vb             varchar2(32767);

begin

 if test_run = 'A'
  then
    for j in 1..run_loop loop
     va := 'X' || to_char(j) || 'Y';
     vb := 'X' || to_char(j) || 'Y';
     if va = vb
      then
       vl := '1';
      else
       vl := '2';
     end if;
    end loop;
   return;
 end if;

 if test_run = 'B'
  then
    for j in 1..run_loop loop
     va := 'A234567890' || to_char(j) || 'Y';
     vb := 'A234567890' || to_char(j) || 'Y';
     if va = vb
      then
       vl := '1';
      else
       vl := '2';
     end if;
    end loop;
   return;
 end if;

 if test_run = 'C'
  then
    for j in 1..run_loop loop
     va := 'A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890' || to_char(j) ||
'Y';
     vb := 'A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890' || to_char(j) ||
'Y';
     if va = vb
      then
       vl := '1';
      else
       vl := '2';
     end if;
    end loop;
   return;
 end if;

 if test_run = 'D'
  then
    for j in 1..run_loop loop
     va :=
'A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2
34567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345
67890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345678
90A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A
234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234
567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567
890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890
A234567890A234567890A234567890A234567890A234567890A234567890' || to_char(j) || 'Y';
     vb :=
'A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2
34567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345
67890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345678
90A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A
234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234
567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567
890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890
A234567890A234567890A234567890A234567890A234567890' || to_char(j) || 'Y';
     if va = vb
      then
       vl := '1';
      else
       vl := '2';
     end if;
    end loop;
   return;
 end if;

 if test_run = 'E'
  then
    for j in 1..run_loop loop
      va :=
'A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2
34567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345
67890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345678
90A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A
234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234
567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567
890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890
A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23
4567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23456
7890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23456789
0A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2
34567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345
67890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345678
90A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A
234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345
     67890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23
```

```
4567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23456
7890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23456789
0A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2
34567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345
67890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345678
90A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A
234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234
567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567
890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890
A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23
4567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23456
7890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23456789
0A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2
34567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345
6789
     0A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567
890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890
A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23
4567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23456
7890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23456789
0A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2
34567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345
67890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345678
90A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A
234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234
567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567
890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890
A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23
4567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23456
7890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23456789
0A23
     4567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A
234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234
567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567
890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890
A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23
4567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23456
7890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23456789
0A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2
34567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345
67890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345678
90A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A
234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234
567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567
890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890
A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23
4567
     890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345
67890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345678
90A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A
234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234
567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567
890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890
A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23
4567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23456
7890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23456789
0A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2
34567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345
67890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345678
90A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A
234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234
567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567
890A
     234567890A234567890A234567890A234567890' || to_char(j) || 'Y';
      vb :=
'A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2
34567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345
67890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345678
90A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A
234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234
567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567
890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890
A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23
4567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23456
7890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23456789
0A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2
34567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345
67890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345678
90A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A
234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345
     67890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23
4567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23456
7890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23456789
0A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2
34567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345
67890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345678
90A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A
234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234
567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567
890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890
A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23
4567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23456
7890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23456789
0A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2
34567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345
6789
     0A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567
890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890
A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23
4567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23456
7890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23456789
0A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2
34567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345
67890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345678
90A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A
234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234
567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567
```

```
890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890
A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23
4567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23456
7890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23456789
0A23
     4567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A
234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234
567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567
890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890
A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23
4567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23456
7890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23456789
0A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2
34567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345
67890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345678
90A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A
234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234
567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567
890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890
A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23
4567
     890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345
67890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345678
90A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A
234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234
567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567
890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890
A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23
4567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23456
7890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A23456789
0A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2
34567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345
67890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A2345678
90A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A
234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234
567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567890A234567
890A
     234567890A234567890A234567890A234567890' || to_char(j) || 'Y';
     if va = vb
      then
       vl := '1';
      else
       vl := '2';
     end if;
    end loop;
   return;
 end if;

end a11;
/
sho err
```

```
create or replace procedure a12(run_loop in integer, test_run in char)
as

 vx             varchar2(100);
 vl             varchar2(100);
 va             varchar2(100);

begin
 --begin htp.init; exception when others then null; end;

 if test_run = 'X'
  then
    for j in 1..run_loop loop
      if mod(j,100000) = 999999 then htp.init; end if;
      htp.p( '<TD>Simple Simon met a pieman going to the fair</TD>' );
    end loop;
   return;
 end if;

 if test_run = 'Y'
  then
    for j in 1..run_loop loop
      if mod(j,100000) = 999999 then htp.init; end if;
      htp.tabledata( 'Simple Simon met a pieman going to the fair' );
    end loop;

   return;
 end if;

end a12;
/
sho err
```

```
create or replace procedure a13(run_loop in integer, test_run in char)
as

 clb            clob;
 Gvx            varchar2(32767);
 vl             varchar2(100);
 va             varchar2(100);

 procedure addtxt( vx varchar2 )
 as
 begin
  begin
   Gvx := Gvx || vx;
  exception
   when others then
    dbms_lob.writeappend( clb, length(Gvx), Gvx );
    Gvx := vx;
  end;
 end addtxt;

begin

 if test_run = 'X'
  then
    for j in 1..run_loop loop
      htp.p( htf.tabledata('Simple Simon met a pieman going to the fair') );
    end loop;
   return;
 end if;

 if test_run = 'Y'
  then
    dbms_lob.createtemporary( clb, TRUE );
    for j in 1..run_loop loop
      addtxt( htf.tabledata('Simple Simon met a pieman going to the fair') );
    end loop;
    dbms_lob.writeappend( clb, length(Gvx), Gvx );
    dbms_lob.freetemporary( clb );
   return;
 end if;

end a13;
/
sho err
```

```
create or replace procedure a14(run_loop integer, test_run in char)
as

 TYPE myarray IS TABLE OF VARCHAR2(100) INDEX BY BINARY_INTEGER;

 cursor c1y( src varchar2) is select pk, colval from d2_temp where colval like '%' || src || '%';

 c1rec          c1y%ROWTYPE;

 vx             myarray;

begin

 if test_run = 'X'
  then

   -- Populate Table
   vx.delete;
   for j in 1..100000 loop

    vx(j) := chr(60+mod(j,100)) || to_char(j) || lpad(chr(60+mod(j,100)),20,'X');

   end loop;

   -- Do Query
   for j in 1..run_loop loop

    for k in vx.first..vx.last loop

     if vx(k) like '%' || to_char(j) || '%' then exit; end if;

    end loop;

   end loop;

 end if;

 if test_run = 'Y'
  then

   -- Populate Table
   delete from d2_temp;
   for j in 1..100000 loop

    insert into d2_temp(pk,colval) values (j, chr(60+mod(j,100)) || to_char(j) || lpad(chr(60+mod(j,100)),20,'X') );

   end loop;

   for j in 1..run_loop loop

    open c1y( to_char(j) );
    fetch c1y into c1rec;
    close c1y;

   end loop;

 end if;

end a14;
/
sho err
```

```
-- Test 15
create or replace package b2a
as

 procedure run_b2a(run_loop integer);

end b2a;
/

create or replace package body b2a
as

 procedure b2a1(p1 in integer, p2 in varchar2, p3 in date);
 procedure b2a2(r1 in integer, r2 in varchar2, r3 in date);
 procedure b2a3(v1 in integer, v2 in varchar2, v3 in date);

 procedure b2a1(p1 in integer, p2 in varchar2, p3 in date)
 as
  vp1 integer;
  vp2 varchar2(1000);
  vp3 date;
 begin
  vp1 := p1 + 100;
  vp2 := p2 || p2;
  vp3 := p3 + 100;
  b2a2(vp1, vp2, vp3 );

 end b2a1;

 procedure b2a2(r1 in integer, r2 in varchar2, r3 in date)
 as
  vp1 integer;
  vp2 varchar2(1000);
  vp3 date;
 begin
  vp1 := r1 - 281;
  vp2 := r2 || r2;
  vp3 := r3 - 645;
  b2a3(vp1, vp2, vp3 );

 end b2a2;

 procedure b2a3(v1 in integer, v2 in varchar2, v3 in date)
 as
  vp1 integer;
  vp2 varchar2(1000);
  vp3 date;
 begin
  vp1 := v1;
  vp2 := v2;
  vp3 := v3;
 end b2a3;

 procedure run_b2a(run_loop integer)
 as
 begin
  for j in 1..run_loop loop

   b2a1(j, to_char(mod(j,100)+60),sysdate + j/100);

  end loop;
 end run_b2a;

end b2a;
/
sho err
```

```
create or replace package b2b
as

 procedure run_b2b(run_loop integer);

end b2b;
/

create or replace package b2b1
as

 procedure b2a1(p1 in integer, p2 in varchar2, p3 in date);

end b2b1;
/

create or replace package b2b2
as

 procedure b2a2(r1 in integer, r2 in varchar2, r3 in date);

end b2b2;
/

create or replace package b2b3
as

 procedure b2a3(v1 in integer, v2 in varchar2, v3 in date);

end b2b3;
/

create or replace package body b2b1
as

 procedure b2a1(p1 in integer, p2 in varchar2, p3 in date)
 as
  vp1 integer;
  vp2 varchar2(1000);
  vp3 date;
 begin
  vp1 := p1 + 100;
  vp2 := p2 || p2;
  vp3 := p3 + 100;
  b2b2.b2a2(vp1, vp2, vp3 );

 end b2a1;

end b2b1;
/

create or replace package body b2b2
as

 procedure b2a2(r1 in integer, r2 in varchar2, r3 in date)
 as
  vp1 integer;
  vp2 varchar2(1000);
  vp3 date;
 begin
  vp1 := r1 - 281;
  vp2 := r2 || r2;
  vp3 := r3 - 645;
  b2b3.b2a3(vp1, vp2, vp3 );

 end b2a2;

end b2b2;
/

create or replace package body b2b3
as

 procedure b2a3(v1 in integer, v2 in varchar2, v3 in date)
 as
  vp1 integer;
  vp2 varchar2(1000);
  vp3 date;
 begin
  vp1 := v1;
  vp2 := v2;
  vp3 := v3;
 end b2a3;

end b2b3;
/

create or replace package body b2b
as

 procedure run_b2b(run_loop integer)
 as
 begin
  for j in 1..run_loop loop

   b2b1.b2a1(j, to_char(mod(j,100)+60),sysdate + j/100);

  end loop;
 end run_b2b;

end b2b;
/
```

```
create or replace procedure a16(run_loop in integer, test_run in char)
as

 MYGLOBAL       clob;
 mylocal        clob;

 procedure subproc( alocalclob in out clob )
 as
  x             integer;
  tmp           varchar2(100);
 begin
  tmp := 'The quick brown fox';
  x := length( tmp );
  dbms_lob.writeappend( alocalclob, x, tmp );
 end subproc;

 procedure subproc_nocopy( blocalclob in out nocopy clob )
 as
  x             integer;
  tmp           varchar2(100);
 begin
  tmp := 'The quick brown fox';
  x := length( tmp );
  dbms_lob.writeappend( blocalclob, x, tmp );
 end subproc_nocopy;

 procedure subproc_nocopy_nested( clocalclob in out nocopy clob, ctr in integer )
 as
  x             integer;
  tmp           varchar2(100);
 begin
  if ctr = 0 then return; end if;
  tmp := 'The quick brown fox';
  x := length( tmp );
  dbms_lob.writeappend( clocalclob, x, tmp );
  subproc_nocopy_nested( clocalclob, ctr - 1);
 end subproc_nocopy_nested;

 procedure subproc_nested( clocalclob in out clob, ctr in integer )
 as
  x             integer;
  tmp           varchar2(100);
 begin
  if ctr = 0 then return; end if;
  tmp := 'The quick brown fox';
  x := length( tmp );
  dbms_lob.writeappend( clocalclob, x, tmp );
  subproc_nested( clocalclob, ctr - 1);
 end subproc_nested;

 procedure global_subproc
 as
  x             integer;
  tmp           varchar2(100);
 begin
  tmp := 'The quick brown fox';
  x := length( tmp );
  dbms_lob.writeappend( MYGLOBAL, x, tmp );
 end global_subproc;

begin
 if test_run = 'A'
  then
    dbms_lob.createtemporary(mylocal, TRUE, DBMS_LOB.call);
    for j in 1..run_loop loop
     subproc( mylocal );
    end loop;
    dbms_lob.freetemporary( mylocal );
   return;
 end if;
 if test_run = 'B'
  then
    dbms_lob.createtemporary(mylocal, TRUE, DBMS_LOB.call);
    for j in 1..run_loop loop
     subproc_nocopy( mylocal );
    end loop;
    dbms_lob.freetemporary( mylocal );
   return;
 end if;
 if test_run = 'C'
  then
    dbms_lob.createtemporary(mylocal, TRUE, DBMS_LOB.call);
    subproc_nocopy_nested( mylocal, run_loop );
    dbms_lob.freetemporary( mylocal );
   return;
 end if;
 if test_run = 'D'
  then
    dbms_lob.createtemporary(mylocal, TRUE, DBMS_LOB.call);
    subproc_nested( mylocal, run_loop );
    dbms_lob.freetemporary( mylocal );
   return;
 end if;
 if test_run = 'E'
  then
    dbms_lob.createtemporary(MYGLOBAL, TRUE, DBMS_LOB.call);
    for j in 1..run_loop loop
     global_subproc;
    end loop;
    dbms_lob.freetemporary( MYGLOBAL );
   return;
 end if;

end a16;
/

 sho err
```

```
create or replace procedure a17(run_loop in integer, test_run in char)
as

 cursor c1(x varchar2) is select count(*) tot from temp_x where col1 = x;
 cursor c2(y varchar2) is select count(*) tot from temp_y where col1 = y;
 cursor c3(z varchar2) is select count(*) tot from temp_z where col1 = z;

 tot integer;

begin

 if test_run = 'X' -- Standard Table
  then
    for j in 1..run_loop loop
     insert into temp_x(col1) values (to_char(60 + mod(j,60)) || systimestamp);
    end loop;
    for j in 1..run_loop loop
     open c1( 'xxxxx' );
     fetch c1 into tot;
     close c1;
    end loop;
    execute immediate 'truncate table temp_x';
   return;
 end if;

 if test_run = 'Y' -- Global Temp Preserve Rows
  then
    for j in 1..run_loop loop
     insert into temp_y(col1) values (to_char(60 + mod(j,60)) || systimestamp);
    end loop;
    for j in 1..run_loop loop
     open c2( 'yyyyy' );
     fetch c2 into tot;
     close c2;
    end loop;
    execute immediate 'truncate table temp_y';
   return;
 end if;

 if test_run = 'Z' -- Global Temp Delete Rows
  then
    for j in 1..run_loop loop
     insert into temp_z(col1) values (to_char(60 + mod(j,60)) || systimestamp);
    end loop;
    for j in 1..run_loop loop
     open c2( 'zzzzz' );
     fetch c2 into tot;
     close c2;
    end loop;
    execute immediate 'truncate table temp_z';
   return;
 end if;

end a17;
/

sho err
```

```
create or replace procedure a18(run_loop in integer, test_run in char)
as

 TYPE myarray      IS TABLE OF VARCHAR2(4000) INDEX BY BINARY_INTEGER;

 x              number;
 y              number;
 z              varchar2(100);
 a              myarray;

begin

 if test_run = 'X'
  then
    for j in 1..run_loop loop
      x := 5 / j;
      y := x * 3.2;
      z := to_char(60 + mod(j,60));
      a(mod(j,60)) := z;
    end loop;
   return;
 end if;

 if test_run = 'Y'
  then
    for j in 1..run_loop loop
     begin

      x := 5 / j;
      y := x * 3.2;
      z := to_char(60 + mod(j,60));
      a(mod(j,60)) := z;

     exception
      when others then x := 1;
     end;

    end loop;
   return;
 end if;

end a18;
/

sho err
```

```
create or replace package a19
as
 procedure main(run_loop in integer, test_run in char);
end a19;
/

create or replace package body a19
as

procedure nested_call( ctr in integer )
as
 x              integer;
 y              varchar2(1000);

 procedure nested_subproc(subctr in integer, subx in integer, suby in varchar2 )
 as
  z             number;
  lvar          varchar2(1000);
 begin
  z := subctr / subx;
  lvar := suby || to_char(50+mod(z,70));
 end nested_subproc;

begin

 x := ctr + 1000;
 y := to_char(60 + mod(ctr,100));
 nested_subproc(ctr, x, y );

end nested_call;



procedure non_nested_subproc(subctr in integer, subx in integer, suby in varchar2 )
as
 z              number;
 lvar           varchar2(1000);
begin
 z := subctr / subx;
 lvar := suby || to_char(50+mod(z,70));
end non_nested_subproc;

procedure non_nested_call( ctr in integer )
as
 x              integer;
 y              varchar2(1000);
begin

 x := ctr + 1000;
 y := to_char(60 + mod(ctr,100));
 non_nested_subproc(ctr, x, y );

end non_nested_call;


procedure main(run_loop in integer, test_run in char)
as

begin

 if test_run = 'X'
  then
    for j in 1..run_loop loop
      nested_call(j);
    end loop;
   return;
 end if;

 if test_run = 'Y'
  then
    for j in 1..run_loop loop
     non_nested_call(j);
    end loop;
   return;
 end if;

end main;

end a19;
/

sho err
```

```
create or replace procedure a20_x(run_loop in integer, test_run in char)
as

 x                             integer;
 y                             integer;
 z                             varchar2(1000);
 dummy1                        clob;
 dummy2                        blob;
 dummy3                        clob;
 dummy4                        blob;
 dummy1a        integer;
 dummy1b        number;
 dummy1c        binary_float;
 dummy1d        pls_integer;
 dummy2v1       varchar2(32767);
 dummy2v2       varchar2(32767);
 dummy2v3       varchar2(32767);
 dummy2v4       varchar2(32767);
 dummy2v5       varchar2(32767);
 dummy1_rec     a5x%ROWTYPE;
 dummy2_rec     a5x%ROWTYPE;
 dummy3_rec     a5x%ROWTYPE;
 dummy4_rec     a5x%ROWTYPE;
 dummy5_rec     a5x%ROWTYPE;
 dummy6_rec     a5x%ROWTYPE;
 dummy7_rec     a5x%ROWTYPE;
 dummy8_rec     a5x%ROWTYPE;
 dummy9_rec     a5x%ROWTYPE;
 dummy10_rec    a5x%ROWTYPE;
 dummy2v6       varchar2(32767);
 dummy2v7       varchar2(32767);
 dummy2v8       varchar2(32767);
 dummy2v9       varchar2(32767);
 dummy2v10      varchar2(32767);
 dummyx1_rec    a5y%ROWTYPE;
 dummyx2_rec    a5y%ROWTYPE;
 dummyx3_rec    a5y%ROWTYPE;
 dummyx4_rec    a5y%ROWTYPE;
 dummyx5_rec    a5y%ROWTYPE;
 dummyx6_rec    a5y%ROWTYPE;
 dummy2v11      varchar2(32767);
 dummy2v12      varchar2(32767);
 dummy2v13      varchar2(32767);
 dummy2v14      varchar2(32767);
 dummy2v15      varchar2(32767);

begin
 -- Declare all variables to ensure the compiler does not reject them because they are not referenced
  x                             := 0;
  y                             := 0;
  z                             := 'xxx';
  dummy1                        := empty_clob;
  dummy2                        := empty_blob;
  dummy3                        := empty_clob;
  dummy4                        := empty_blob;
  dummy1a                       := 0;
  dummy1b                       := 0;
  dummy1c                       := 0;
  dummy1d                       := 0;
  dummy2v1      := lpad( '+', 32000, '-' );
  dummy2v2      := lpad( '+', 32000, '-' );
  dummy2v3      := lpad( '+', 32000, '-' );
  dummy2v4      := lpad( '+', 32000, '-' );
  dummy2v5      := lpad( '+', 32000, '-' );
  dummy1_rec    := null;
  dummy2_rec    := null;
  dummy3_rec    := null;
  dummy4_rec    := null;
  dummy5_rec    := null;
  dummy6_rec    := null;
  dummy7_rec    := null;
  dummy8_rec    := null;
  dummy9_rec    := null;
  dummy10_rec   := null;
  dummy2v6      := lpad( '+', 32000, '-' );
  dummy2v7      := lpad( '+', 32000, '-' );
  dummy2v8      := lpad( '+', 32000, '-' );
  dummy2v9      := lpad( '+', 32000, '-' );
  dummy2v10     := lpad( '+', 32000, '-' );
  dummyx1_rec.col_y := lpad( '+', 1000, '-' );
  dummyx2_rec.col_y := lpad( '+', 1000, '-' );
  dummyx3_rec.col_y := lpad( '+', 1000, '-' );
  dummyx4_rec.col_y := lpad( '+', 1000, '-' );
  dummyx5_rec.col_y := lpad( '+', 1000, '-' );
  dummyx6_rec.col_y := lpad( '+', 1000, '-' );
  dummy2v11     := lpad( '+', 32000, '-' );
  dummy2v12     := lpad( '+', 32000, '-' );
  dummy2v13     := lpad( '+', 32000, '-' );
  dummy2v14     := lpad( '+', 32000, '-' );
  dummy2v15     := lpad( '+', 32000, '-' );
 for j in 1..run_loop loop
  x := x + y;
  z := substr(z || to_char(60+mod(j,40)) || to_char(60+mod(x,40)), 1, 800 );
 end loop;

end a20_x;
/

create or replace procedure a20_y(run_loop in integer, test_run in char)
as
 x                             integer;
 y                             integer;
 z                             varchar2(1000);
 dummy1                        clob;
 dummy2                        blob;
 dummy3                        clob;
 dummy4                        blob;
```

```
 dummy1a        integer;
 dummy1b        number;
 dummy1c        binary_float;
 dummy1d        pls_integer;
 dummy2v1       varchar2(32767);
 dummy2v2       varchar2(32767);
 dummy2v3       varchar2(32767);
 dummy2v4       varchar2(32767);
 dummy2v5       varchar2(32767);
 dummy1_rec     a5x%ROWTYPE;
 dummy2_rec     a5x%ROWTYPE;
 dummy3_rec     a5x%ROWTYPE;
 dummy4_rec     a5x%ROWTYPE;
 dummy5_rec     a5x%ROWTYPE;
 dummy6_rec     a5x%ROWTYPE;
 dummy7_rec     a5x%ROWTYPE;
 dummy8_rec     a5x%ROWTYPE;
 dummy9_rec     a5x%ROWTYPE;
 dummy10_rec    a5x%ROWTYPE;
 dummy2v6       varchar2(32767);
 dummy2v7       varchar2(32767);
 dummy2v8       varchar2(32767);
 dummy2v9       varchar2(32767);
 dummy2v10      varchar2(32767);
 dummyx1_rec    a5y%ROWTYPE;
 dummyx2_rec    a5y%ROWTYPE;
 dummyx3_rec    a5y%ROWTYPE;
 dummyx4_rec    a5y%ROWTYPE;
 dummyx5_rec    a5y%ROWTYPE;
 dummyx6_rec    a5y%ROWTYPE;
 dummy2v11      varchar2(32767);
 dummy2v12      varchar2(32767);
 dummy2v13      varchar2(32767);
 dummy2v14      varchar2(32767);
 dummy2v15      varchar2(32767);

begin
 -- Declare all variables to ensure the compiler does not reject them because they are not referenced
  x                           := 0;
  dummy1                      := empty_clob;
  dummy2                      := empty_blob;
  dummy3                      := empty_clob;
  dummy4                      := empty_blob;
  dummy1a                     := 0;
  dummy1b                     := 0;
  dummy1c                     := 0;
  dummy1d                     := 0;
  dummy2v1      := lpad( '+', 32000, '-' );
  dummy2v2      := lpad( '+', 32000, '-' );
  dummy2v3      := lpad( '+', 32000, '-' );
  dummy2v4      := lpad( '+', 32000, '-' );
  dummy2v5      := lpad( '+', 32000, '-' );
  z                           := 'xxx';
  dummy1_rec    := null;
  dummy2_rec    := null;
  dummy3_rec    := null;
  dummy4_rec    := null;
  dummy5_rec    := null;
  dummy6_rec    := null;
  dummy7_rec    := null;
  dummy8_rec    := null;
  dummy9_rec    := null;
  dummy10_rec   := null;
  dummy2v6      := lpad( '+', 32000, '-' );
  dummy2v7      := lpad( '+', 32000, '-' );
  dummy2v8      := lpad( '+', 32000, '-' );
  dummy2v9      := lpad( '+', 32000, '-' );
  dummy2v10     := lpad( '+', 32000, '-' );
  dummyx1_rec.col_y := lpad( '+', 1000, '-' );
  dummyx2_rec.col_y := lpad( '+', 1000, '-' );
  dummyx3_rec.col_y := lpad( '+', 1000, '-' );
  dummyx4_rec.col_y := lpad( '+', 1000, '-' );
  dummyx5_rec.col_y := lpad( '+', 1000, '-' );
  dummyx6_rec.col_y := lpad( '+', 1000, '-' );
  dummy2v11     := lpad( '+', 32000, '-' );
  dummy2v12     := lpad( '+', 32000, '-' );
  dummy2v13     := lpad( '+', 32000, '-' );
  dummy2v14     := lpad( '+', 32000, '-' );
  dummy2v15     := lpad( '+', 32000, '-' );
  y                           := 0;

 for j in 1..run_loop loop
  x := x + y;
  z := substr(z || to_char(60+mod(j,40)) || to_char(60+mod(x,40)), 1, 800 );
 end loop;

end a20_y;
/

sho err
```