

PLSQL. Too easy for its own good?

Erik van Roon

Who Am I?

Erik van Roon






- Originally analyst of microbiology and biochemistry
- Oracle developer since 1995, self-employed since 2009



EvROCS
COMPLETING THE PUZZLE

- Most of my work takes place inside the database
- Did several successful major datamigration projects
- Core member of the MASH program



 : erik.van.roon@evrocs.nl
 : www.evrocs.nl
 : [@evrocs_nl](https://twitter.com/evrocs_nl)



Mentor and Speaker Hub

Our goal is to *connect* speakers with mentors to assist in *preparing* technical sessions and *improving* presentation skills

Interested? Read more and get in touch

<https://mashprogram.wordpress.com>

What did I walk into.....



A client with minimal (PL)SQL skills

Who built a company critical DB application on (PL)SQL



Tables
Primary Keys

About 300
On every table
Often all columns except 1 or 2
Often including column "creation date"

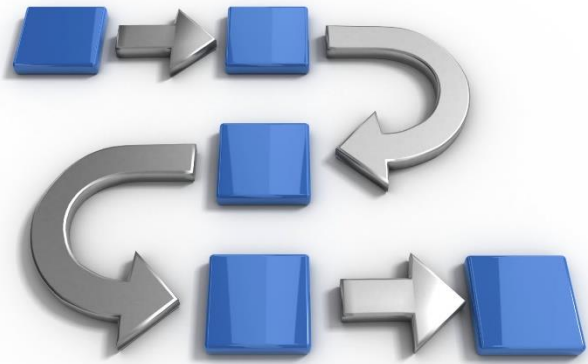
Foreign Keys
Unique constraints
Check constraints
Triggers

< 10
Zero
Zero
< 10

Data integrity.....



Deploying Code



VBA Tool
Hobby project

Deploying anything
~ 10 minutes

Even to Dev DB
deploy tool had to be used



The tool changed the code

- Removed duplicate spaces/tabs
- Removed Empty lines
- Ensured a / to run each statement

For a package:

Add / on newline after

```
end;
```

Or

```
end name;
```

Introduce 21st century:

```
case x  
  when y  
    then do_y;  
  else do_z;  
end case;
```

Fix the symptom, forget about the cause
...and about the consequences...

Deploy to database is slow and problematic???



Whenever possible, put the functionality in script files
Put them on the filesystem (no deploy tool)
Run them from SQL*Plus
Create a complex chain of shell scripts to

Start SQL*Plus
Initialize (values to parameter tables)
Execute or schedule a script
Catch errors and output
Redirect errors and output to appropriate directories



Fix the symptom, forget about the cause ...and about the consequences...



Email comes in...

From : DBA
To : everybody
Subject : Check queries

Hi, there are 12 check queries that have now been running for a week in prod.
Usually they only take 5 days.

Can I abort them?

My thoughts...

What ??

"Only 5 days"??

On this database??

I would be ashamed to deploy a query that runs for an hour!!

Let's have a look and fix the queries!!!

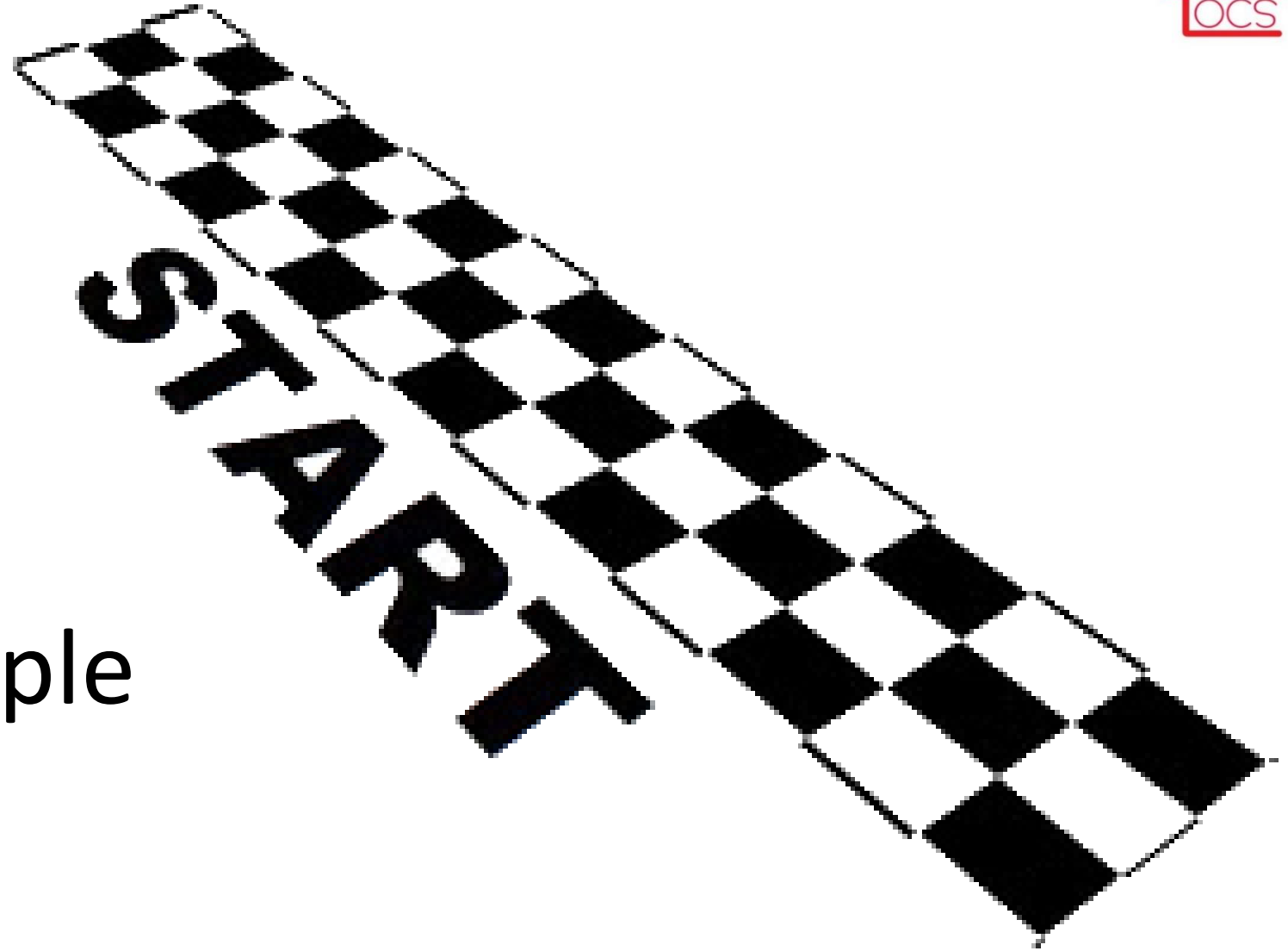
Thoughts of the developer...

From : Guy responsible
To : everybody
Subject : Re: Check queries

Hi, we can speed them up by running for only 1% of the data.



Let's start simple



Constantly variable



```
procedure do_something  
Is
```

```
    const_value
```

```
    varchar2(50) := 'My Constant Value';
```

```
begin
```

```
    ...
```



Constantly variable

```
procedure do_something  
Is
```

```
    const_value constant varchar2(50) := 'My Constant Value';
```

```
begin  
    ...
```

That's better



Deafening



Silence

The Invisible Man



```
begin
  ...
  begin
    delete
    from   my_table
    where  id = l_id;
  exception
    when  others
    then
      null;
  end;
  ...
end;
```



I really don't want to see the error that can't happen

Better safe



Than sorry

Don't delete if you don't have to....



```
begin
```

Delete a row

```
delete  
from parameters_table  
where id = l_id;
```

```
end;
```

Don't delete if you don't have to....



But first check if it exists

```
begin
  select count(*)
  into    l_count
  from    parameters_table
  where   id = l_id;
```

Delete a row

```
if l_count > 0
then
  delete
  from    parameters_table
  where   id = l_id;
end if;
```

```
end;
```


Don't delete if you don't have to....

But first check if it exists

```
begin
  select count(*)
  into   l_count
  from   parameters_table
  where  id = l_id;
```

Delete a row

```
if l_count > 0
then
  delete
  from   parameters_table
  where  id = l_id;
end if;
```

And give a useless message on error

```
exception
  when others
  then
    raise_application_error
      (-20000, 'Unknown Error in procedure abc');
end;
```

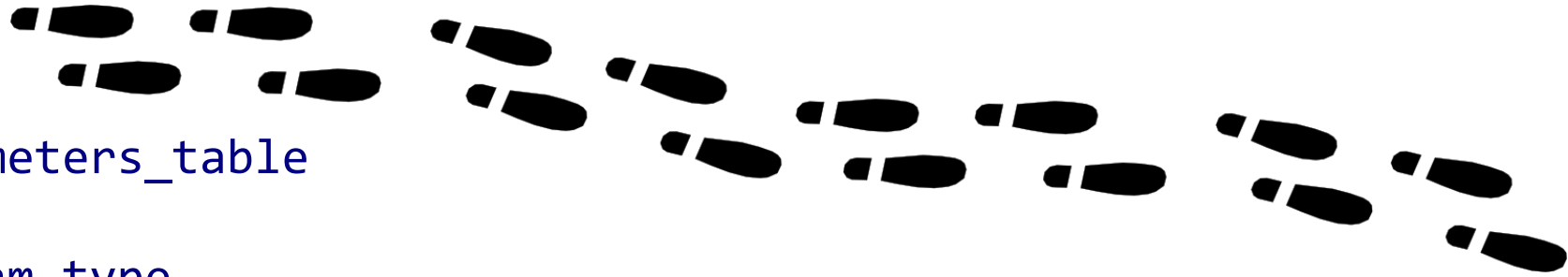


Work.....



You can never have enough of it

Don't do in one step what you can do in two...



```
begin
  insert
  into parameters_table
    (id
     ,param_type
     ,param_value
    )
  values (p_id
         ,p_param_type
         ,p_param_value
        );
```

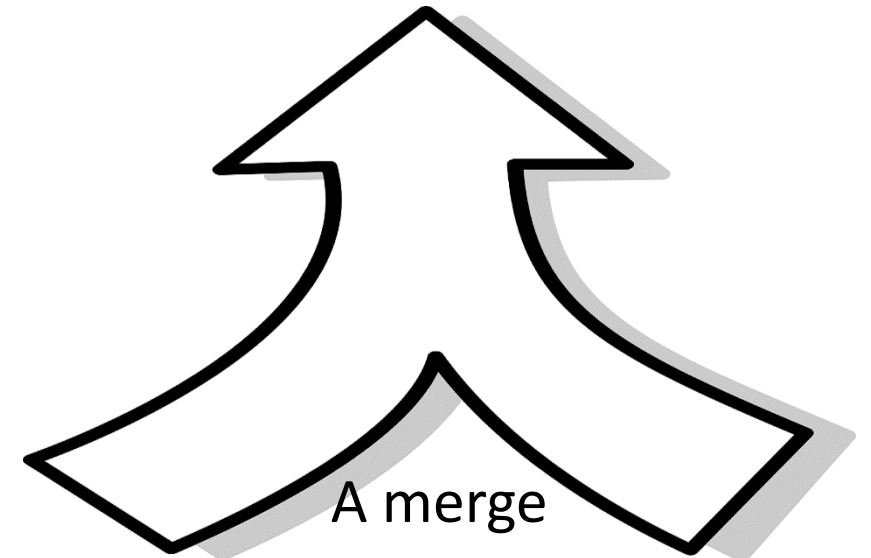
```
end;
```

Don't do in one step what you can do in two...

```
begin
  insert
  into   parameters_table
        (id
         ,param_type
         ,param_value
        )
  values (p_id
         ,p_param_type
         ,p_param_value
        );
exception
  when dup_val_on_index
  then
    update parameters_table
    set   param_type = p_param_type
    ,    param_value= p_param_value
    where id         = p_id
end;
```



Should have been



If only Calculation



could be done easily

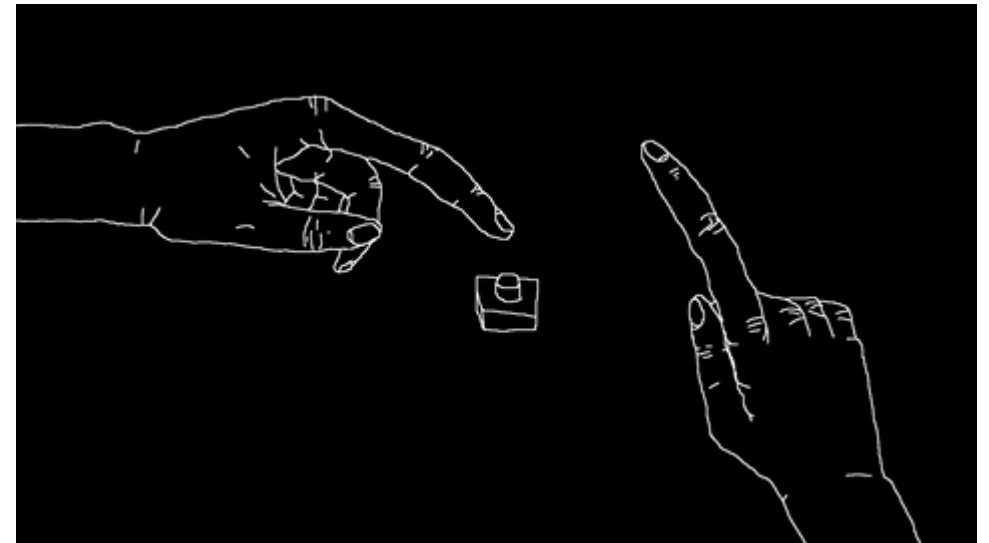
Calculation in PLSQL is so hard...

```

declare
  l_start          date;
  l_end            date;
  l_runtime_in_minutes number;
begin
  select sysdate
  into   l_start
  from   dual;
  ...
  select sysdate
  into   l_end
  from   dual;

  select (l_end - l_start) * 24 * 60
  into   l_runtime_in_minutes
  from   dual;
  ...
end;
```

You can never have enough context switches



Really, so hard to do calculations...

```
declare
  l_start          date;
  l_runtime_in_minutes number;
begin
  l_start := sysdate;
  ...
  l_runtime_in_minutes := (sysdate - l_start) * 24 * 60;
  ...
end;
```

And so slow...

```
Number of iterations          : 1.000.000
Selecting from dual          , Elapsed seconds: 44,53
Using PLSQL functionality, Elapsed seconds: 0,74

PL/SQL procedure successfully completed.
```

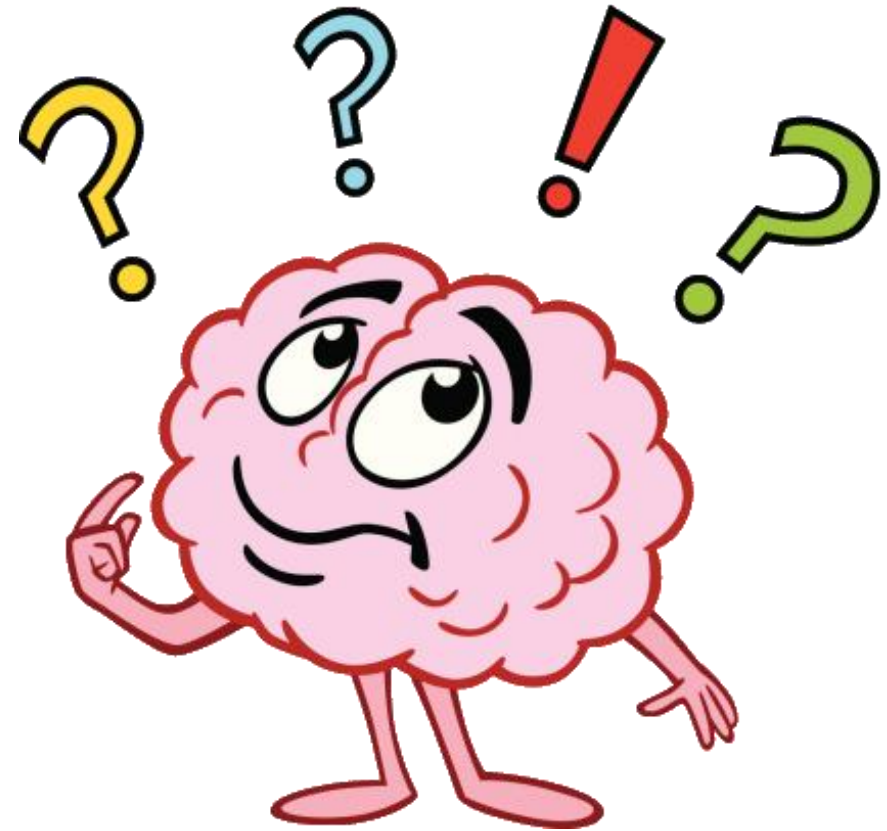
The selects from dual
took **60** times as long

Missing

Data

If only there were ways to deal with NULL

```
procedure do_something
(p_input in varchar2
)
is
  l_input varchar2(4000);
begin
  if p_input is null
  then
    l_input := 'Default Input';
  else
    l_input := p_input;
  end if;
  ...
end;
```



```
l_input := nvl (p_input, 'Default Input');
```

```
-- OR
```

```
l_input := coalesce (p_input, 'Default Input');
```

Reading everything



Reading until the last chapter



```
l_line := '===File Is Empty===';  
while l_line is not null  
loop  
begin  
    utl_file.get_line(l_handle, l_line);  
exception  
when no_data_found  
then  
    l_line := null;  
end;  
  
if l_line is not null  
then  
    do_something_with(l_line);  
end if;  
end loop;
```

Right
There



Also 'last chapter' might be skipped

```
"FIRST_COLUMN", "SECOND_COLUMN", "THIRD_COLUMN"  
"First row", 123, "1963-03-12"  
"Second row", 456, "1966-07-03"  
"Third row", 789, "1996-05-25"  
  
"Fifth row", 234, "1934-01-17"  
"Sixth row", 567, "2021-09-11"
```

Right
There

Counting



Manually

Can't trust Oracle to be able to count



```
begin
  fetch my_cursor
  bulk collect
  into my_collection;

  l_idx := my_cursor.first;
  while l_idx is not null
  loop
    l_row_count := l_row_count + 1;
    l_idx := my_cursor.next (l_idx);
  end loop;

  -- from now on use l_row_count
  -- for the size of the collection
  ...
end;
```





When your date

isn't

what you expected

Why does Oracle sometimes ignore time in dates?

What do you mean? It doesn't!

Yes it does!! Look here, this script.....

...Wait, why: alter session set nls_date_format = 'dd-mm-yyyy';

Well, so to_date() will work correctly of course, duh!

Why not a format mask in your to_date()??

A what ???

Anyway, not important. We pass it a date, not a string.

Okay!?! Let's have a look then.....



See, a date + time goes in, but date without time gets into the table

```
procedure write2table
(p_startdate in date
)
is
begin
  insert
  into my_table
  (date_column
  )
  values (to_date(p_startdate)
  );
end;
```



Do you have any idea what is happening here?
I mean, any at all?

Caching values

To save work



Procedure should

- Receive a **name** and a **value**
- If name cached before: update value
- Otherwise cache name and value

What my first thought would be...

Declare global associative array

- Name **g_array**
- Datatype **varchar2(...)** <= Value
- Indexed by **varchar2(...)** <= Name

```
procedure cache_it
(p_name in varchar2
,p_value in varchar2
)
is
begin
    g_array(p_name) := p_value;
end;
```

What I found...

Declare global associative array

- Name **g_array**
- Datatype **record (Name + Value)**
- Indexed by **pls_integer**

Loop through array....

....but why use a for loop if you have while?

```
l_idx := g_array.first;  
while l_idx is not null  
loop
```

```
g_array.next(l_idx);
```

```
end loop;
```

What I found...

Declare global associative array

- Name **g_array**
- Datatype **record (Name + Value)**
- Indexed by **pls_integer**

Check if row is for the name
to be cached

If not, check next row

```
l_idx := g_array.first;
while l_idx is not null
loop
    if g_array(l_idx).name = p_name
    then

        else
            g_array.next(l_idx);
        end if;
    end loop;
```

What I found...

Declare global associative array

- Name **g_array**
- Datatype **record (Name + Value)**
- Indexed by **pls_integer**

If it is: cache the value here

```
l_idx := g_array.first;
while l_idx is not null
loop
    if g_array(l_idx).name = p_name
    then
        g_array(l_idx).value := p_value;

    else
        g_array.next(l_idx);
    end if;
end loop;
```

What I found...

Declare global associative array

- Name **g_array**
- Datatype **record (Name + Value)**
- Indexed by **pls_integer**

...And skip the rest of the array

```
l_idx := g_array.first;
while l_idx is not null
loop
    if g_array(l_idx).name = p_name
    then
        g_array(l_idx).value := p_value;
        l_idx                := null;

    else
        g_array.next(l_idx);
    end if;
end loop;
```

What I found...

Declare global associative array

- Name **g_array**
- Datatype **record (Name + Value)**
- Indexed by **pls_integer**

Oh yes, and remember that we found the name

```
l_idx := g_array.first;
while l_idx is not null
loop
    if g_array(l_idx).name = p_name
    then
        g_array(l_idx).value := p_value;
        l_idx                 := null;
        l_updated              := true;
    else
        g_array.next(l_idx);
    end if;
end loop;
```

What I found...

Declare global associative array

- Name **g_array**
- Datatype **record (Name + Value)**
- Indexed by **pls_integer**

If name not found

Increase global variable containing
highest index number

Cache name and value

```
l_idx := g_array.first;
while l_idx is not null
loop
    if g_array(l_idx).name = p_name
    then
        g_array(l_idx).value := p_value;
        l_idx                := null;
        l_updated            := true;
    else
        g_array.next(l_idx);
    end if;
end loop;

if not l_updated
then
    g_max_idx := g_max_idx + 1;
    ga_setting(g_max_idx).name := p_name;
    ga_setting(g_max_idx).value := p_value;
end if;
```




```

l_idx := g_array.first;
while l_idx is not null
loop
  if g_array(l_idx).name = p_name
  then
    g_array(l_idx).value := p_value;
    l_idx                := null;
    l_updated            := true;
  else
    g_array.next(l_idx);
  end if;
end loop;

if not l_updated
then
  g_max_idx := g_max_idx + 1;
  ga_setting(g_max_idx).name := p_name;
  ga_setting(g_max_idx).value := p_value;
end if;

```

**Taking it
one step
at a time**

```
for r_parent in (select id
                 from parent
                 where ...)
loop
  l_parent_id := r_parent.id;
```

```
end loop;
```

PL/SQL =

Procedural

Language extension to
SQL





PL/SQL =

Procedural

Language extension to
SQL

```
for r_parent in (select id
                 from   parent
                 where  ...)
```

```
loop
  l_parent_id := r_parent.id;
```

```
for r_child in (select id, col_1, col_2
                from   child
                where  child.parent_id = l_parent_id)
```

```
loop
  l_child_id      := r_child.id;
  l_child_col_1   := r_child.col_1;
  l_child_col_2   := r_child.col_2;
```

```
end loop
```

```
end loop;
```



PL/SQL =

Procedural

Language extension to
SQL

```
for r_parent in (select id
                  from   parent
                  where  ... )
loop
  l_parent_id := r_parent.id;

  for r_child in (select id, col_1, col_2
                  from   child
                  where  child.parent_id = l_parent_id)
  loop
    l_child_id      := r_child.id;
    l_child_col_1   := r_child.col_1;
    l_child_col_2   := r_child.col_2;

    insert
    into   target
          (id           , parent_id  , child_id  , col_1   , col_2   )
    values (targetseq.nextval, l_parent_id, l_child_id, l_col_1, l_col_2);
  end loop
end loop;
```

Because this isn't
Procedural

Why Not.....

```

insert
into    target
        (id            , parent_id  , child_id  , col_1   , col_2   )
select targetseq.nextval , p.id      , c.id      , c.col_1 , c.col_2
from    parent    p
join    child      c
      on  c.parent_id = p.id
where   ...
;

```



Similar



But not the same

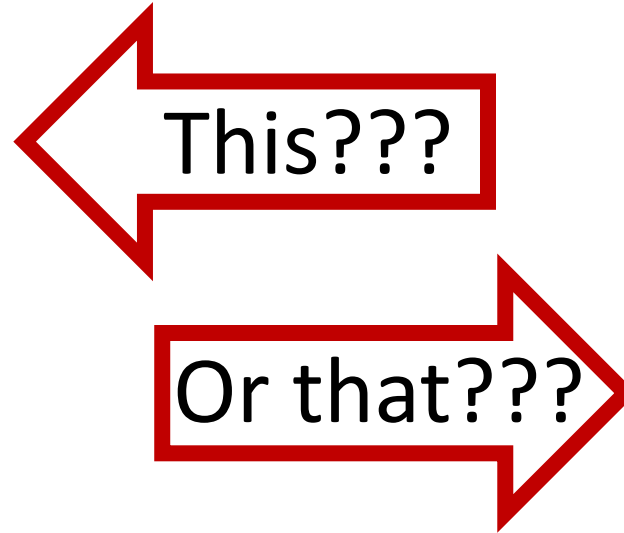
```
procedure do_it
is

begin
  for idx in 1 .. 100
  loop
    ...
    other_proc (idx);
    ...
  end loop;
  ....

end;
```

Good syntax

"idx" implicitly declared



```
procedure do_it
is
  idx pls_integer;
begin
  for idx in 1 .. 100
  loop
    ...
    other_proc (idx);
    ...
  end loop;
  ....

end;
```

Good syntax

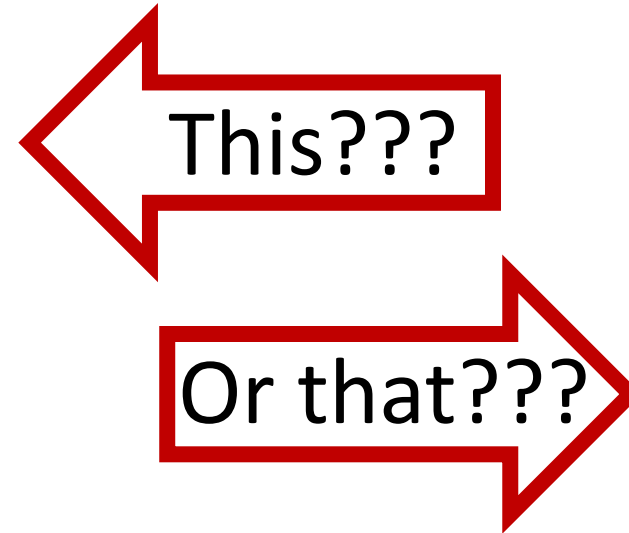
"idx" explicitly declared

So?? What's the problem


```
procedure do_it
is
begin
  for idx in 1 .. 100
  loop
    ...
    other_proc (idx);
    ...
  end loop;
  ....
  yet_another_proc (idx);
end;
```

Warning about mistake

Code will NOT compile



```
procedure do_it
is
  idx pls_integer;
begin
  for idx in 1 .. 100
  loop
    ...
    other_proc (idx);
    ...
  end loop;
  ....
  yet_another_proc (idx);
end;
```

Code WILL compile with bug.

2nd procedure gets null-value

Implicit idx is different variable than Explicit idx

Usefull

Useless

**Treating the pain
Instead of the wound**

utl_file sometimes misses line endings !!!

What do you mean? It doesn't!

Yes it does!! Look here, this file

```
USER_ID;USER_NAME  
101;K. Böhmer  
102;P. Stænsma  
103;T. Munro Peña  
104;R. Merçan  
105;C. Bākhuis  
106;J. Weintré  
107;S. Årssen  
108;V. Ortye
```

It sees 103+104 as one row and 106+107

But I fixed that...

What's left is that the funny characters still get malformed



The applied 'Fix'....

```
--Inside loop until end of file (see funny solution before)  
utl_file.get_line (l_myfile, l_line, 32767);  
l_count := l_count + 1;
```

```
l_array (l_count) := l_line;
```

Original, before fix: Put every line in an array

The applied 'Fix'....

```
--Inside loop until end of file (see funny solution before)
utl_file.get_line (l_myfile, l_line, 32767);
l_count := l_count + 1;

while instr(l_line , chr(10)) > 0
loop

end loop;

l_array (l_count) := l_line;
```

Loop as long as we find end-of-lines

The applied 'Fix'....

```
--Inside loop until end of file (see funny solution before)
utl_file.get_line (l_myfile, l_line, 32767);
l_count := l_count + 1;

while instr(l_line , chr(10)) > 0
loop
    l_array (l_count) := substr(l_line, 1, instr(l_line , chr(10)) - 1);
    l_line           := substr(l_line, instr(l_line , chr(10)) + 1);
    l_count          := l_count + 1;
end loop;

l_array (l_count) := l_line;
```

Strip first part of the 'line' and put in array

The REAL solution



INVESTIGATE !!!!



The cause



UTF-8 Characterset
Multibyte



Windows-1252 Encoding
Singlebyte

UTL_FILE Expects same characteraset as database

In UTF-8...

A byte starting with	Indicates
0	1 byte character
110	First byte of 2 byte character
1110	First byte of 3 byte character
11110	First byte of 4 byte character

103;T. Munro Peña
104;R. Merçan

Character	In Windows 1252		In UTF-8	
	Hex	Binary	Indicates	Combines in 1 chr
ñ	F1	11110001	4-byte chr	ñ + a + [eol] + 1

106;J. Weintré
107;S. Årsen

Character	In Windows 1252		In UTF-8	
	Hex	Binary	Indicates	Combines in 1 chr
é	E9	11101001	3-byte chr	é + [eol] + 1

So, use something that supports other Character Set

```
with my_file as
  (select rownum as line_nr, text_line
   from external
     ((rn integer, text_line varchar2(400))
    type oracle_loader
    default directory MY_INPUTFILE_DIR
    access parameters
      (records
        delimited by newline
        character set WE8MSWIN1252
        nologfile nobadfile nodiscardfile
        fields
          missing field values are null
          (rn recnum, text_line char(400))
        )
      location ('FileToRead.csv')
      reject limit unlimited
    )
  )
select rn as line_nr , text_line
from my_file;
```

One possible solution:

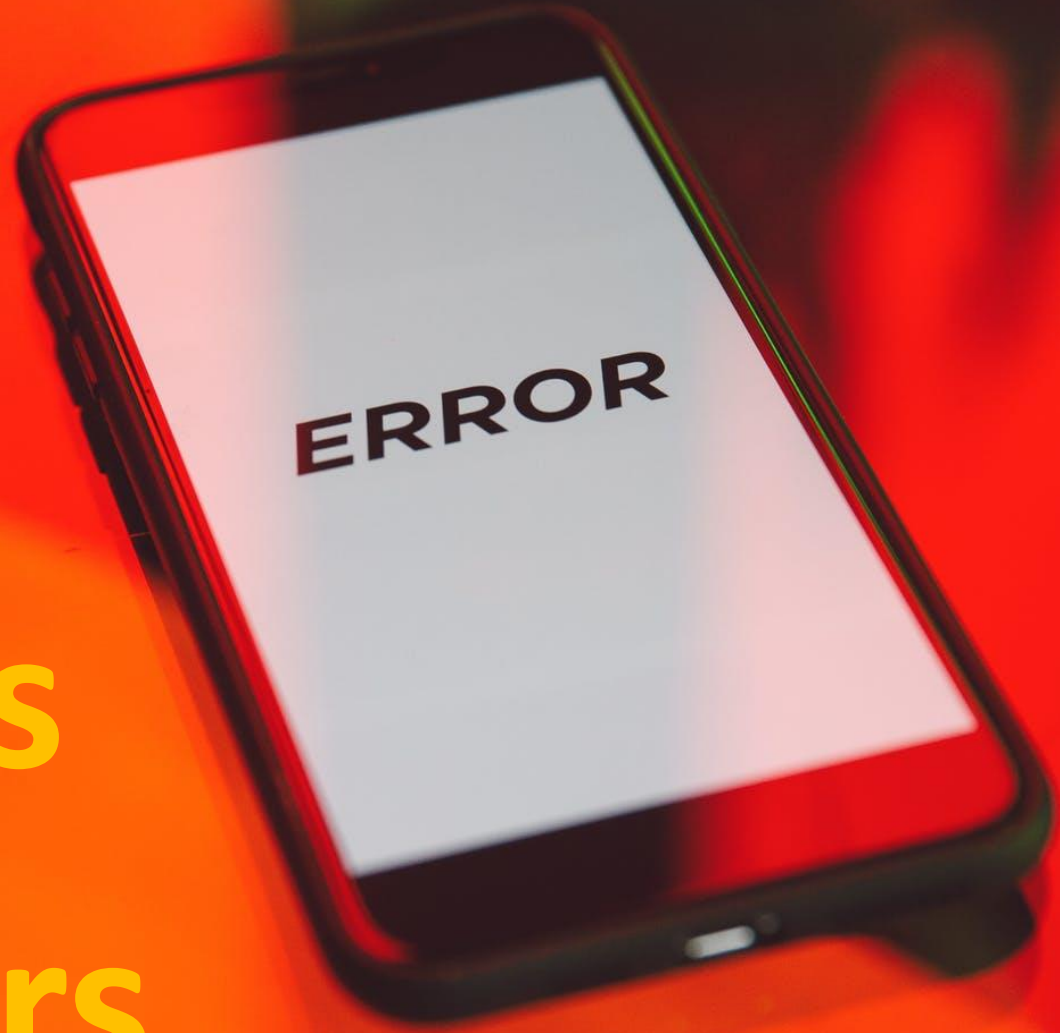
Inline external table ($\geq 18c$)
($< 18c$: normal external table)

You can set the character set

If Dynamic SQL: as a parameter

All other External Table features
available: split csv in fields

Bulk collect the query and done



Erroneous Errors

Oracle produces nonsense error messages !!!

What do you mean? It doesn't!

Yes it does!! The error messages have nothing to do with the actual error

**Look: I raise ORA-20000
Oracle reports ORA-00032: invalid session migration password**

Hmmm. Worrying!!

Let's have a look then.....



The shell scripts used to run the code does:

-
- Starts SQL*Plus
- Runs the PLSQL script
 - PLSQL script returns errorcode as exit code
- Determines the error message of the code
- Logs the code and message
-

Understandable because even Oracle docs say:

The commands in the following script cause SQL*Plus to exit and return the SQL error code if the SQL UPDATE command fails:

```
WHENEVER SQLERROR EXIT SQL.SQLCODE
UPDATE EMP_DETAILS_VIEW SET SALARY = SALARY*1.1;
```

And it does, but.....

Oracle Error message range

Currently (21c): **0 - 65535**

Exit code range

Linux & windows: **0 - 255**

Remember?



Look: I raise **ORA-20000**
Oracle reports **ORA-00032**: invalid session migration password



$\text{MOD}(20000, 256) = ???$

32



“Stupid questions do exist.

But it takes a lot more time and energy to correct a stupid mistake than it takes to answer a stupid question, so please ask your stupid questions.”

a wise teacher who taught me more than just physics

Thanks !!!