ITOUG
ITALIAN ORACLE USER GROUP

ITOUG Tech Day 2026

Developers are like cats
They don't give a ....

PL/SQL antipatterns.

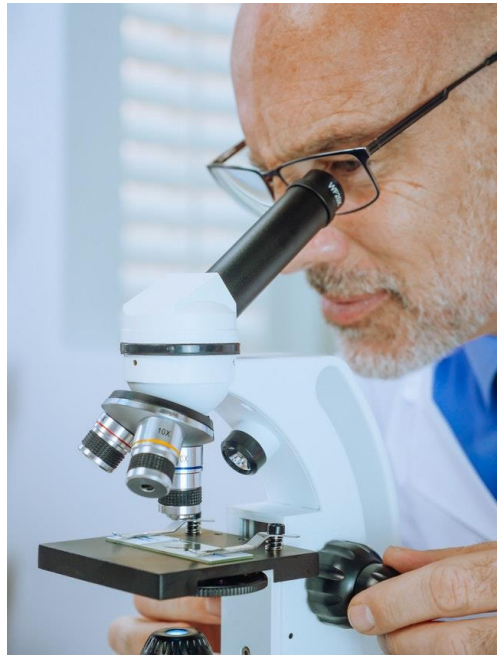
Erik van Roon

EvROCS
COMPLETING THE PUZZLE


FITNESS CLUB
PISCINA DELLE ROSE

Who Am I?

Erik van Roon



1995

ORACLE DATABASE
>=Oracle5

2009

EvROCS
COMPLETING THE PUZZLE

SYM42
Member of Symposium 42
https://sym42.org/

MASH
MASH Program

Oracle ACE Pro

: erik.van.roon@evrocs.nl
: www.evrocs.nl
: ~~@evrocs_nl~~
: @evrocs.bsky.social

MASH PROGRAM

Mentor and Speaker Hub

BECOME A SPEAKER - MENTORING - EVOLVING - IMPROVING

.SYMPOSIUM 42

Created by the community, to support the community

Sharing of reliable knowledge
Supporting the various user groups and individuals

@sym_42

@sym42.bsky.social

https://sym42.org/

Oracle ACE

Tech Superstars Unite

Get worldwide recognition as an Oracle ACE

Oracle.com profile page

Swag, certification exam credit & event passes

Exclusive content

Networking events

Your own Oracle cloud account

Travel support

@oracleace

Linkedin.com/groups/72183

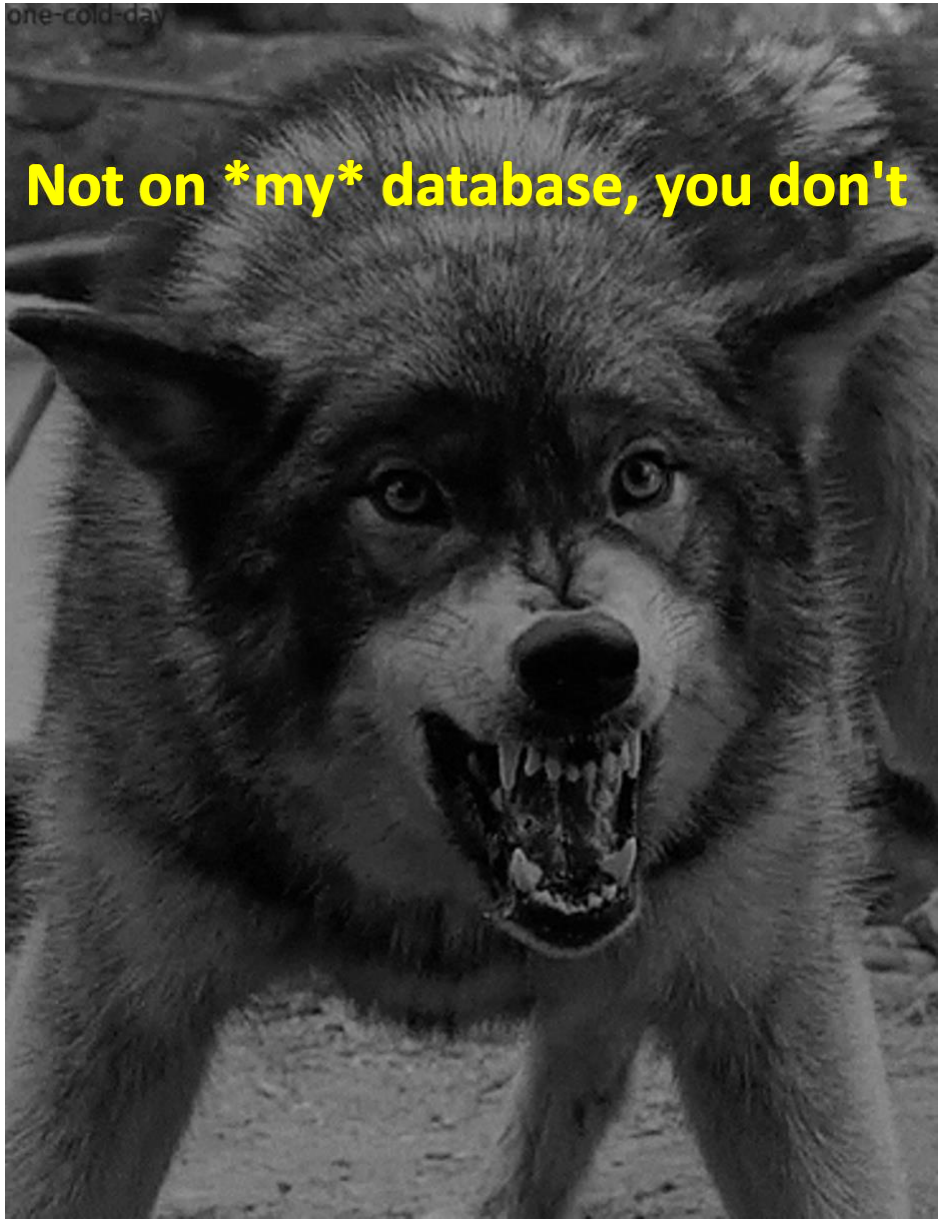
@oracleace.bsky.social

Learn more at:
ace.oracle.com

So? We're cats, right?

So, DBA's are….?

"We need this change"



Anti-Pattern

WIKIPEDIA

"An **anti-pattern** is any common but counterproductive solution to some class of problem."

CATegory

Hey, I've just started learning this shit

- A constant that is declared as a variable
- A variable that is initialised as null
- A loop iterator that is explicitly declared (no it's not)
- Start with closing a local cursor if it's open
- Relying on default for the parameter to never be null

```
create or replace procedure just_starting
(p_must_have_value  in  number   default 1
)
is
  cursor c_dual
  is
  select *
  from   dual;

  cn_will_never_change   varchar2(30)   := 'Constant value';

  l_var_1   integer := null;
  l_index   integer;
begin
  if c_dual%isopen
  then
    close c_dual;
  end if;

  for l_index in 1 .. 1000
  loop
    some_procedure(l_index + p_must_have_value);
  end loop;

  [...]
end;
```

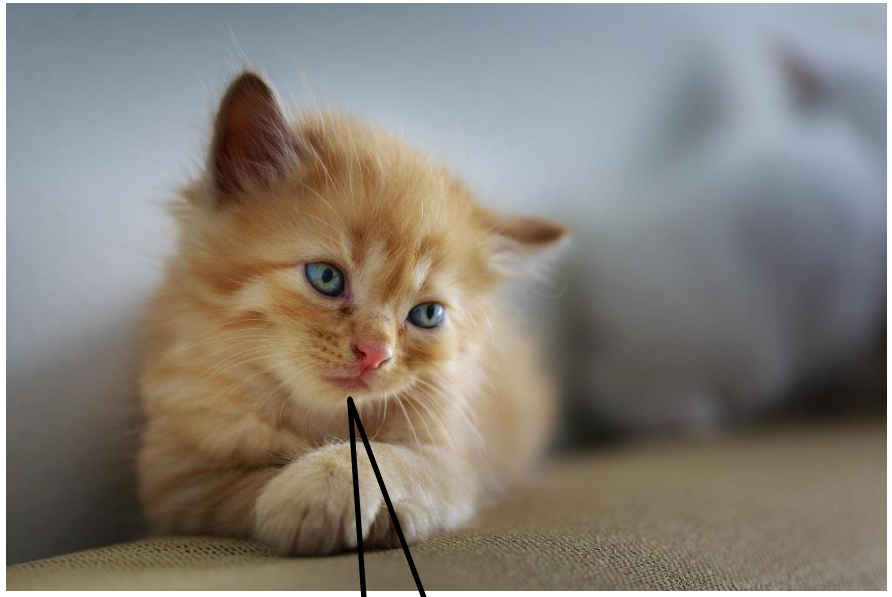
Raising the wrong exception….

```
declare
  l_dummy   dual.dummy%type;
  no_data_found exception;
begin
  select dummy
  into   l_dummy
  from   dual
  where  1 = 2
  ;
exception
  when no_data_found
  then
    dbms_output.put_line ('Nothing found!');
  when others
  then
    dbms_output.put_line ('OMG, something went
terribly wrong: '||sqlerrm);
end;
```

Userdefined exception is handled in exception handling
But it's not the one raised by the "select … into"

```
14    when others
15    then
16      dbms_output.put_line ('OMG, something went terribly wrong: '||sqlerrm);
17  end;
18* /
OMG, something went terribly wrong: ORA-01403: no data found
```

Never just say "It's better to …."
Always explain exactly why!



```
select distinct main.factory_code
from   (
         select eqt.factory_code
         from   app_objects       obj
         ,      app_equipments    urg
         where  eqt.id           = obj.urg_id
           and  obj.cag_id       = :b_cag_id
           and  obj.nr_volg      = :b_cot_nr_volg
           and  eqt.factory_code is not null
         union all
         select pet.factory_code
         from   app_objects       obj
         ,      app_packages      pet
         where  pet.id           = obj.pet_id
           and  obj.cag_id       = :b_cag_id
           and  obj.nr_volg      = :b_cot_nr_volg
           and  pet.factory_code is not null
       ) main
```

Oh??
Ok, I guess. If you say so.

Nice. But uhmmm
It's better to use union all

CATegory

Some cats just want to watch the world burn

Write code as if the person who must maintain it is a psychopath who knows where you live

Should counters be 0-based or 1-based?

Oracle employees….

…. Let's be inclusive to all opinions.

In a Headstart (for Designer) library for Oracle Forms:

```
v_errorrec.msg_text := REPLACE (v_errorrec.msg_text, '<p1>'
v_errorrec.msg_text := REPLACE (v_errorrec.msg_text, '<p2>'
v_errorrec.msg_text := REPLACE (v_errorrec.msg_text, '<p3>'
v_errorrec.msg_text := REPLACE (v_errorrec.msg_text, '<p4>'
rrec.msg_text := REPLACE (v_errorrec.msg_text, '<p5>'
rrec.msg_text := REPLACE (v_errorrec.msg_text, '<p6>'
rrec.msg_text := REPLACE (v_errorrec.msg_text, '<p7>'
rrec.msg_text := REPLACE (v_errorrec.msg_text, '<p8>'
rrec.msg_text := REPLACE (v_errorrec.msg_text, '<p9>'
rrec.msg_text := REPLACE (v_errorrec.msg_text, '<p10>'
```

How about ANSI joins or Oracle syntax joins?

Whichever you prefer, use them as intended

```
cursor c_001 (b_ctt_nr  ***.nr%type)
is
  with
  w_ckp as
  (
    select cut.lot_id   lot_id
    from   ***    cut
    ,      ***    ckp
    where  cut.ctt_nr   = ckp.ctt_nr_1
      and  ckp.ctt_nr_2 = b_ctt_nr
  )
  , w_cot as
  (
    select cot.lot_id   lot_id
    from   ***    ctt
    ,      ***    cot
    where  cot._nr      = ctt.cot_nr
      and  ctt.nr       = b_ctt_nr
  )
select coalesce (ckp.lot_id, cot.lot_id) lot_id
from            w_ckp    ckp
full outer join w_cot    cot

;
```

Any guesses
about the resultset?

Table_1

	T1_NR
1	1
2	2
3	3
4	4
5	5

Table_2

	T2_NR
1	3
2	4
3	5
4	6
5	7

```
select t1.t1_nr
,      t2.t2_nr
from             table_1  t1
full outer join table_2  t2
  on    1 = 1
order by t1.t1_nr
,        t2.t2_nr
;
```

=

```
select t1.t1_nr
,      t2.t2_nr
from       table_1  t1
cross join table_2  t2
order by t1.t1_nr
,        t2.t2_nr
;
```

=

```
select t1.t1_nr
,      t2.t2_nr
from table_1  t1
,    table_2  t2
order by t1.t1_nr
,        t2.t2_nr
;
```

	T1_NR	T2_NR
1	1	3
2	1	4
3	1	5
4	1	6
5	1	7
6	2	3
7	2	4
8	2	5
9	2	6
10	2	7
11	3	3
12	3	4
13	3	5
14	3	6
15	3	7
16	4	3
17	4	4
18	4	5
19	4	6
20	4	7
21	5	3
22	5	4
23	5	5
24	5	6
25	5	7

Becomes a cross join 🤔

Turns out: This was the intention 😯

One of the advantages of ANSI joins…

Makes your intentions clear

🤣

It's essential to comment your code….

….not to turn your code into comments

```
create or replace package [illegible] is
  type t_debug_levels is record
  ( none              varchar2(10) := 'NONE'
  , error             varchar2(10) := 'ERR'
  , info              varchar2(10) := 'INFO'
  , all_msg           varchar2(10) := 'ALL'
  );

  c_debug_levels t_debug_levels;

/*  procedure log_message( p_type_log           in
                       , p_database_proc      in  ws
                       , p_database_proc_par  in  ws
                       , p_database_msg_code  in  ws
                       , p_database_msg_text  in  ws
                       , p_debug_text         in  ws
                       , p_xml_data           in  ws
                       );

  procedure log_information( p_database_proc      in
                           , p_database_proc_par  in
                           , p_debug_text         in
                           , p_xml_data           in
                           ) ;
*/

END [illegible];
```

```
create or replace package body [illegible] is
/*
function must_write_log(p_type_log          i
return boolean
is
  c_procedure_name [illegible]
  c_procedure_par [illegible]
            'p_type_log       = ' || p_type_l

  l_melding [illegible]
  l_return boolean := false;
begin

  -- kijk of er logging moet worden aangemaakt
  case [illegible]
    when [illegible]
```

```
end log_information;

begin
  -- Initialization
  [illegible]
*/
begin
  null;
end [illegible];
```

Even if you may want to reverse it one day

You may want to learn about version/source control

Hide complexity in procedures/functions….

….don't create them to introduce complexity

```
function check_date
(p_date    in  date
,p_string  in  varchar2)
return boolean is
  v_return boolean;
begin
  if p_string = '>'
  then
    if p_date  >  trunc(sysdate)
    then v_return := TRUE;
    else v_return := FALSE;
    end if;

  elsif p_string = '<'
  then
    if p_date  <  trunc(sysdate)
    then v_return := TRUE;
    else v_return := FALSE;
    end if;
  [... 3 more elsif ...]
  else
    v_return := FALSE;
  end if;
  return v_return;
end;
```

Function has no added value whatsoever

This: `if check_date(date '2030-07-12' , '>=')`

Is no better than this: `if date '2030-07-12' >= trunc(sysdate)`

But even if you insist on having a function, make it simple:

```
function check_date
(p_date    in  date
,p_string  in  varchar2)
return boolean is
  v_return boolean;
  v_today  date    := trunc(sysdate);
begin
  v_return := (   ( p_string = '>'  and p_date >  v_today )
                or ( p_string = '<'  and p_date <  v_today )
                or ( p_string = '>=' and p_date >= v_today )
                or ( p_string = '<=' and p_date <= v_today )
                or ( p_string =  '=' and p_date  = v_today )
               );
  end if;

  return v_return;
end;
```

CATegory

I'm just as stupid

I built a logging solution….

For various reasons 1 log action can result in multiple log records in a table
So, I prepare the records in a collection
And then……

```
forall i_line in indices of p_msg_lines
  insert
    into ero_logging
         (message
         ,wrapped_line
         ,creation_time
         ,cleanup_time
         ,type_code
         ,codeblock_path
         )
  values (p_msg_lines(i_line).message_line      --> message
         ,p_msg_lines(i_line).wrapped_line      --> wrapped_line
         ,p_msg_lines(i_line).creation_time     --> creation_time
         ,p_msg_lines(i_line).cleanup_time      --> cleanup_time
         ,p_msg_lines(i_line).type_code         --> type_code
         ,p_msg_lines(i_line).codeblock_path    --> codeblock_path
         )
  ;
```

Awesome! Nice and efficiënt!
I'm so good!

Until someone does….

```
  [...]
  forall i in 1 .. a_check.count save exceptions
    insert into ero_check (value)
    values (a_check(i).value);
exception
  when e_bulk_errors
  then
    for i_err in 1 .. sql%bulk_exceptions.count
    loop
      -- log the error
      ero_log.debug_msg
        ('Error '||sql%bulk_exceptions(i_err).error_code);
      -- handle the error
      --........
    end loop;
end;
```

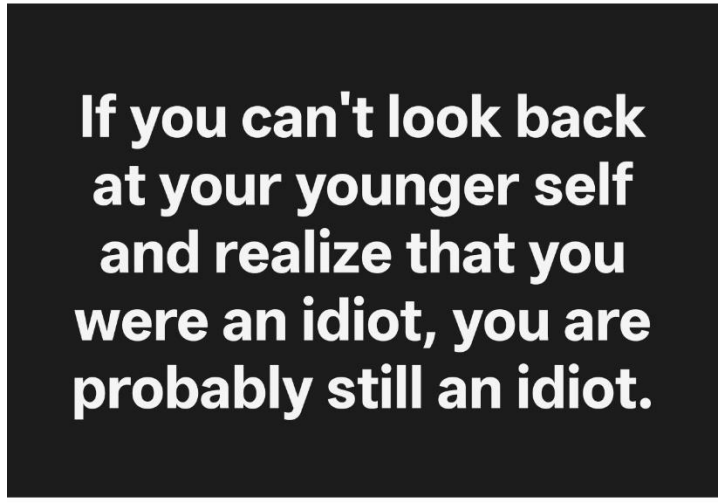
```
declare
*
ERROR at line 1:
ORA-06532: subscript outside of limit
ORA-06512: at line 22
ORA-24381: error(s) in array DML
ORA-06512: at line 13
```

Why???

https://docs.oracle.com/en/database/oracle/oracle-database/19/lnpls/bulk-sql-and-bulk-binding.html#GUID-DAF46F06-EF3F-4B1A-A518-5238B80C69FA

"**Note**
After a FORALL statement *without* the SAVE EXCEPTIONS clause raises an exception, SQL%BULK_EXCEPTIONS.COUNT = 1."

So, even though no "save exceptions" in the logging, SQL%BULK_EXCEPTIONS gets re-initialised



CATegory

Triggers are evil

NO!! Triggers are NOT evil….

….people who misuse/abuse them are evil

```
select count(*)
from   user_triggers;
```

	COUNT(*)
1	7.224

```
select count(*)  as nr_of_lines
from   user_source
where  type = 'TRIGGER'
group by name
order by count(*) desc
;
```

	NR_OF_LINES
1	1.609
2	1.573
3	1.545
4	1.393
5	1.389
6	1.299
7	1.070
8	1.038

And then triggers on table A
Doing DML on table B, C and D
Each with triggers
Doing DML on each other, and A, and X
X has triggers doing DML on Y and Z
Which have triggers doing DML on C and D
…… etc.

So people invent stuff like ⇨

```
create or replace trigger table_a_bri
before insert
on table_a
for each row
declare
begin
  --[......]
  if not my_safety_package.g_table_a_bri_active
  then
    my_safety_package.g_table_a_bri_active := true;

    update table_b
    set    col_1   = :new.col_1
    where  id      = :new.b_id;

    --[......]
  end if;
  --[......]
end;
```

NOOOO! DON'T!!!

How many triggers can you have for 1 event on 1 table?....

....Lots. More than you should

```
create or replace trigger table_a_bri
before insert on table_a for each row
declare
begin
  --[......]
end;
```

```
create or replace trigger table_a_before_row_insert
before insert on table_a for each row
declare
begin
  --[......]
end;
```

```
create or replace trigger table_a_before_row_insert
before insert on table_a for each row follows
table_a_bri
declare
begin
  --[......]
end;
```

Which will fire first?

As they are:

There is no way to tell.

Could be in different order each time.

Can use ordering clause (Follows/Precedes)

But why? I mean..... why?

Even if the order now is irrelevant....

Will it be after 5 years of maintenance?

CATegory

Me, myself and I

Running into a deadlock is bad, ….

….and worse If yours is the only active session in the database

```
create or replace procedure ero_process_data
is
  l_data_id  integer;
begin
  -- [all sorts of processing]

  ero_write_initial_data
    (p_value => 'Whatever'
    ,p_id    => l_data_id
    );

  -- [Do all sorts of checks]

  update ero_important_data
  set    value  = value || ', checked'
  where  id     = l_data_id
  ;

  ero_set_data_status_final (l_data_id);
end;
```

```
exec ero_process_data
```

```
ERO@EVROCS>exec ero_process_data
BEGIN ero_process_data; END;
*
ERROR at line 1:
ORA-00060: deadlock detected while waiting for resource
ORA-06512: at "ERO.ERO_SET_DATA_STATUS_FINAL", line 7
ORA-06512: at "ERO.ERO_PROCESS_DATA", line 19
ORA-06512: at line 1
Help: https://docs.oracle.com/error-help/db/ora-00060/
```

How did I get myself into a deadlock??

Simple, somebody thought this was a good idea

```
create or replace procedure ero_write_initial_data
(p_value  in      ero_important_data.value%type
,p_id        out  ero_important_data.id%type
)
is
  pragma autonomous_transaction;
begin
  insert
  into   ero_important_data
         (created
         ,status
         ,value
         )
  values (sysdate  --> created
         ,'I'      --> status
         ,p_value  --> value
         )
  returning id into p_id
  ;

  commit;
end;
```

```
create or replace procedure ero_set_data_status_final
(p_id      in  ero_important_data.id%type
)
is
  pragma autonomous_transaction;
begin
  update ero_important_data
  set    status   = 'F'
  where  id       = p_id
  ;

  commit;
end;
```

Autonomous transaction is the same session
But a different transaction

So, while process is waiting
for the autonomous transaction
The autonomous transaction is waitig
for the process to commit

CATegory

Back to the future

What if you like to test stuff on a specific sysdate….

….use a timemachine

You could do: `ALTER SYSTEM SET FIXED_DATE =` but you take everybody with you

So, they did….

```
function app$sysdate
return date
is
  l_date_offset  number;
  l_return       date;
begin
  l_return      := sysdate;
  l_date_offset := app_parameter('DATE_OFFSET',user);

  if l_date_offset is not null
  then
    l_return := l_return + l_date_offset;
  end if;

  return l_return;
end;
```

And then…. they changed their mind

```
function app$sysdate
return date
is
  l_date_offset  number;
  l_return       date;
begin
  l_return      := sysdate;
/*  l_date_offset := app_parameter('DATE_OFFSET',u

  if l_date_offset is not null
  then
    l_return := l_return + l_date_offset;
  end if;
*/
  return l_return;
end;
```

Easier (=cheaper) than fixing all the code that uses it

But the application still has all the downsides all over the place

```
cursor c_bca
is
select bcr.contract_period
,      bcr.amount_year
,      apppck.change ('EURO',bcr.currency,bcr.amount_1,app$sysdate)  amount_1
,      apppck.change ('EURO',bcr.currency,bcr.amount_2,app$sysdate)  amount_2
,      apppck.change ('EURO',bcr.currency,bcr.amount_3,app$sysdate)  amount_3
,      apppck.change ('EURO',bcr.currency,bcr.amount_4,app$sysdate)  amount_4
from   app_bulk_contract_rows       bcr
,      app_contract_items           cim
where  cim.id             = bcr.cim_id
  and  cim.contract_date between trunc(app$sysdate) - 7
                         and     trunc(app$sysdate)
;
```

- Always many unnecessary context switches, just so you can test once in a while
- Each execution in the query above returns a different sysdate
- If executed around midnight the selection range may be 8 days instead of 7

CATegory

Giving up too soon

When you need to read a file....

....But don't know when to stop

```
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

And then your file has empty rows

```
"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

CATegory

Having bad character

What if utl_file misses end of lines….

….fix the symptom, not the problem

```
--Inside loop until end of file
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

What if utl_file misses end of lines….

….fix the symptom, not the problem

```
--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

What if utl_file misses end of lines….

….fix the symptom, not the problem

```
--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 problem….

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. Ãrssen
```

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
                  characterset 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 (>= 18c)
(<18c: normal external table)

You can set the characterset
If Dynamic SQL: as a parameter

All other External Table features available: split csv in fields

Bulk collect the query and done

CATegory

Believing the docs

"If it's not in the documentation it isn't true"....

....So if it *is* in the documentation it *must* be true

SQL*Plus User's Guide and Reference says (Still for 26ai):

https://docs.oracle.com/en/database/oracle/oracle-database/26/sqpug/WHENEVER-SQLERROR.html

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, sort of.....

Customer had lots of shell scripts that

- Start SQL*Plus
- Run a PLSQL script
  - PLSQL script returns errorcode as exit code
- Log the error

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

Oracle Error code range

Currently (26ai): **0 - 65535**

MOD (20000 , 256) =	32

Exit code range

Linux & windows: **0 - 255**

NOTE:
After doing this presentation, I have learned that the claim that Windows also has the 0-255 limits (1 byte/8 bit value) is incorrect.
I re-tested it in Windows 11 and noticed that Oracle error codes are correctly passed as exit codes in Windows.
Investigation into when Windows started supporting larger exit codes taught me that it was Windows NT 3.1 (1993), being the first real 32bit Windows, which started using 4 byte/32 bit exit codes (0 to 4294967295). Though I did see this behaviour in Windows 10/Oracle 12c around 2020.
It looks as if (though I don't have proof for that!!!) Oracle first kept the behavior of the Windows-port of SQLPLUS identical to that of the Linux version which had no other option than to use 1 byte exit codes. But that at some point the restriction was lifted for the Windows version.

Also note that, the documentation for "whenever sqlerror" (see link on previous slide) indeed just states that this should work, the documentation for "Exit" (see https://docs.oracle.com/en/database/oracle/oracle-database/26/sqpug/EXIT.html) states that:

> "The range of operating system return codes is also restricted on some operating systems. This limits the portability of EXIT n and EXIT variable between platforms. For example, on UNIX there is only one byte of storage for return codes; therefore, the range for return codes is limited to zero to 255. "

Thanks Daniel Overby Hansen, for pointing this out!!

“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 !!