Ketzerische Gedanken zu SQL und PLSQL

glaub nicht alles was die Experten sagen

Sven-Uwe Weller

syntegris information solutions

Sven-Uwe Weller

- ✓ Syntegris CEO, CTO "Design and Development"
- ✓ Oracle Certified Professional, Oracle Certified Expert, Oracle Ace
- ✓ active OTN Member, Apex, SQL, PLSQL

ORACLE ACE

Mail: sven.weller@syntegris.de
Twitter: @SvenWOracle
Blog: svenweller.wordpress.com

www.syntegris.de

DOGMAS

when OTHERS then null is a BUG

select * is BAD

SQL is always faster than PLSQL

SEQUENCES can not be GAPLESS

DOGMAS

when OTHERS then null is a BUG

Connor McDonald
21. Aug 2017

Hints and Tips - The simple guide to WHEN OTHERS THEN NULL

https://youtu.be/Dw0qRw8P0cQ

Tom Kyte
12. Aug 2001

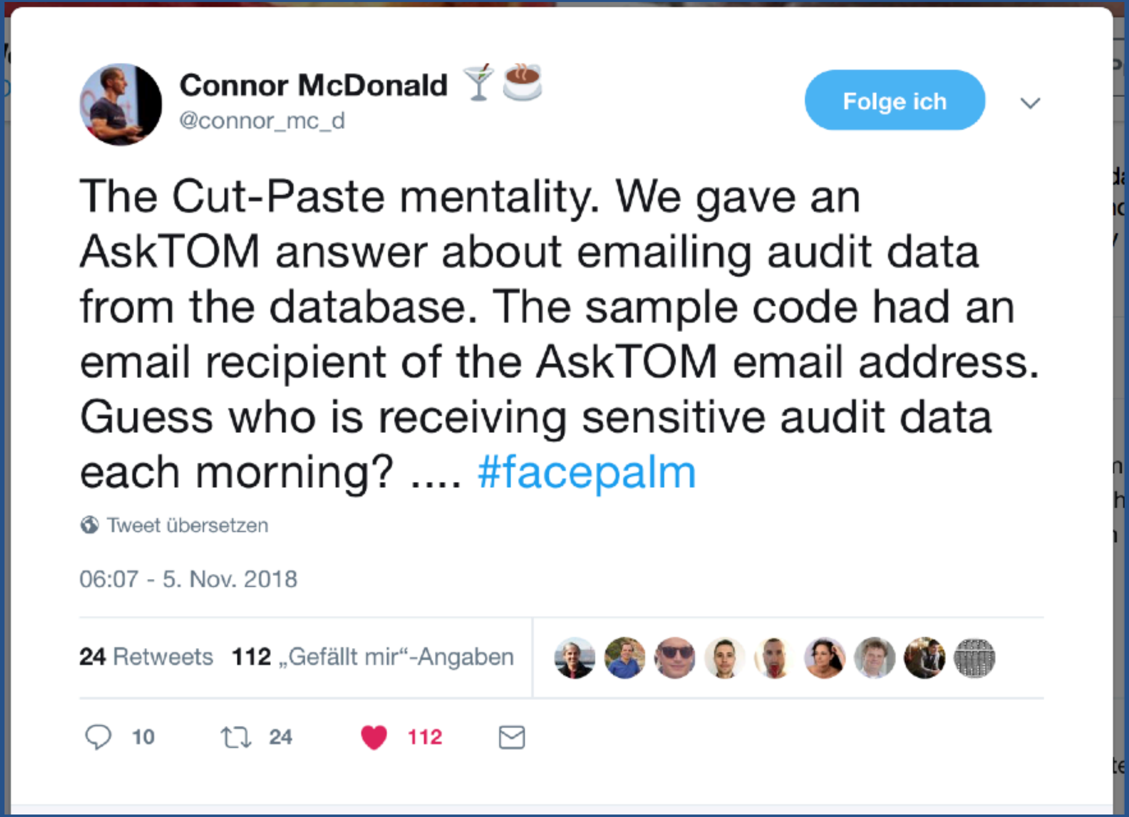
A when others is almost always a BUG unless it is immediately followed by a RAISE.

when OTHERS then null is a BUG

Connors typical target audience?

Wer ist die Zielgruppe?

Gehört Ihr dazu?



Connor McDonald 🍸☕
@connor_mc_d

The Cut-Paste mentality. We gave an AskTOM answer about emailing audit data from the database. The sample code had an email recipient of the AskTOM email address. Guess who is receiving sensitive audit data each morning? .... #facepalm

Tweet übersetzen

06:07 - 5. Nov. 2018

24 Retweets 112 „Gefällt mir"-Angaben

10 24 112

when OTHERS then null is a BUG

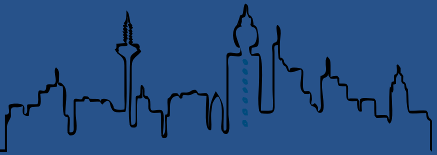
unless

there is a very good reason!

Comment the reason!

Explain why you break the rule in a comment.

sequences can not be gapless



DOGMAS

Tom Kyte
12. Nov 2002

*There is a fact about sequences - an UNDENIABLE, UNESCAPABLE fact - they are **not gap free, will never be gap free, they will have gaps!***

Toon Koppelaars
1. Oct 2009

Q: ... is there a standard technique for avoiding or accounting for gaps?

A: No. And everything you "build-yourself", will be flawed (buggy and/or causing serialization points you do not want).

ORACLE
MY ORACLE SUPPORT

Doc ID 197212.1
15. May 2018

ORACLE
E-BUSINESS SUITE

How To Setup Gapless Document Sequencing in Receivables

Please note that in Oracle Receivables GAPLESS document sequencing only applies to INVOICES. You can use document sequences to uniquely number Receipts, Bills Receivables, Adjustments, and other data objects, but they are *not* guaranteed to be gapless. The implementation steps detailed in this document only applies to Invoices.

SEQUENCES can not be GAPLESS

"sequence" ?

the word "sequence" is used for different things
- the number generator
- the number value
- the stored values in ID column

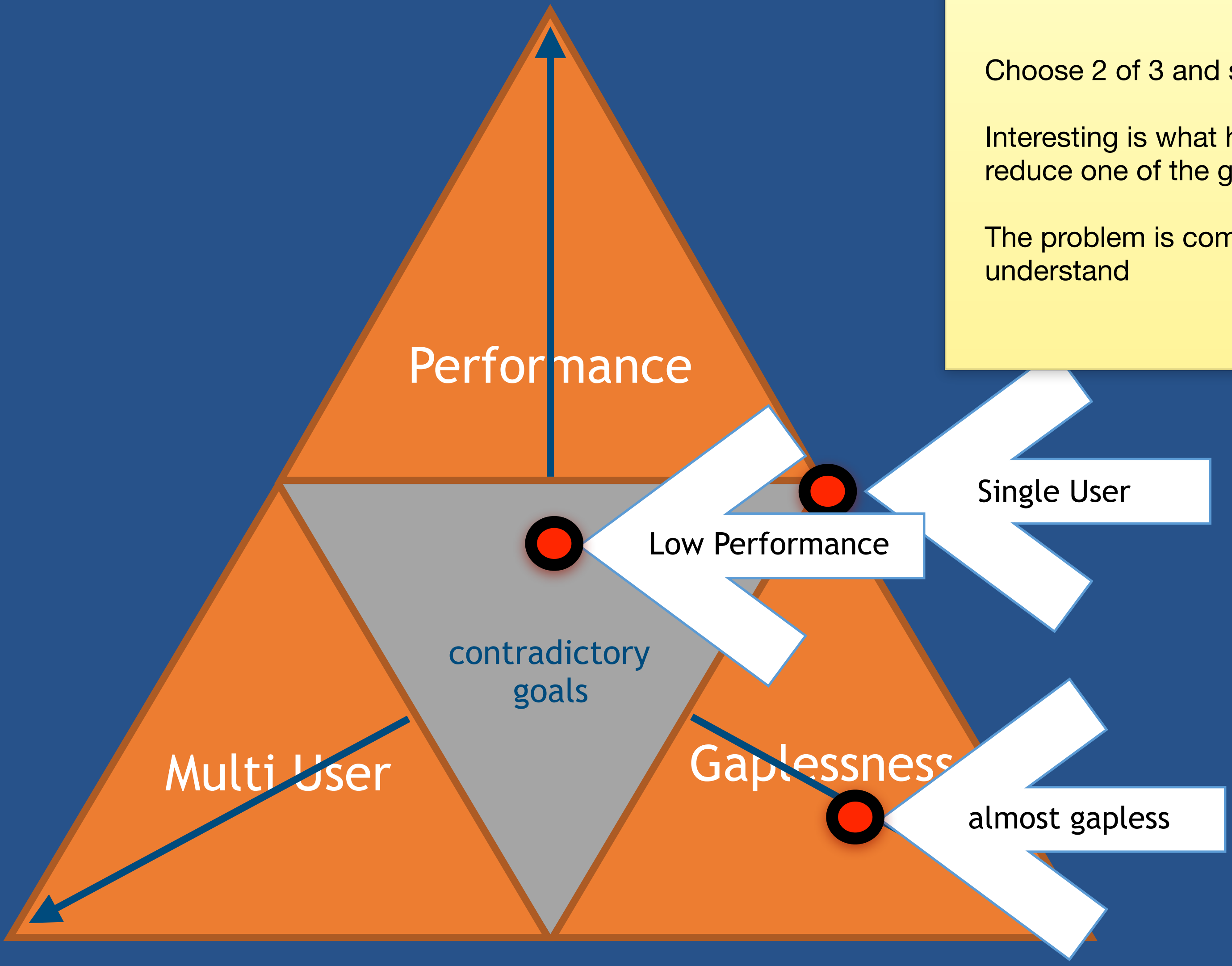
SEQUENCES can not be GAPLESS

"sequence" ?

XY problem

SEQUENCES can not be GAPLESS

complex technical problem



Choose 2 of 3 and sacrifice the other.

Interesting is what happens when we reduce one of the goals.

The problem is complex to understand

SEQUENCES can not be GAPLESS

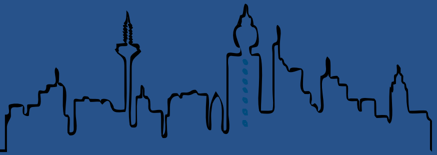
"sequence" ?

XY problem

cognitive bias
and premature closure

WARNING
NO SWIMMING, PLAYING, OR FISHING IN WATER
BEWARE OF DANGEROUS WILDLIFE

select * is bad



DOGMAS

Jonathan Lewis
22. Jul 2015

Obviously you shouldn't use the lazy "" notation in any production code – it can cause several different problems* (including the dangers of *"whoops, I didn't mean to make that one invisible"*)

Jeff Smith
23. Nov 2016

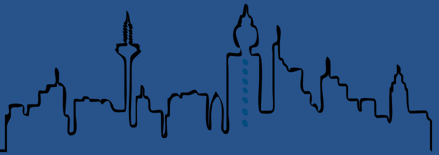
7 ways to avoid

*SELECT * queries in SQL Developer*

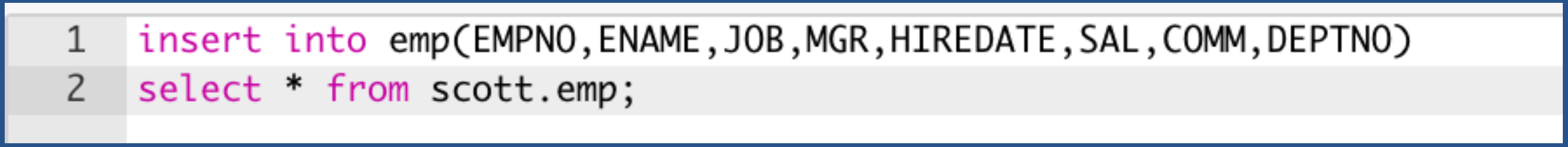
But wait, what's wrong with SELECT * FROM queries?

- you don't need all the columns
- columns can change
- columns can be added
- columns can be removed

At some point, your application (or report) will 'break.'



select * is BAD



```sql
insert into emp(EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO)
select * from scott.emp;
```

this code is bad.

WHY?

1) data redundancy (this code is too simplified. Real cases are way more complex)
2) fixed column order mapped to dynamic column order
   => implicit mapping

select * is BAD

Too often select * is categorized as evil. I think those cases are extremly rare. In most cases the problem is somewhere else. and we should concentrate on the real issues.

freezes column order

table elimination

hidden * expansions

select * is BAD

```
declare
  cursor c_sourcedata
  is (select * -- "generic" column list
      from (
         -- begin dummy select - replace with your own data so
         select level as nr , substr( to_char(to_date('1','J')
         from dual
         connect by level &amp;amp;lt; 1000
        -- end of dummy select
         ) t
      );
```

extract

```
procedure storeResults(p_targetdata in out nocopy targetdata
is
begin
    -- do a bulk insert/update/merge
    forall f in 1..p_index
      insert into testdummy
      values p_targetdata(f);
      -- if needed handle exceptions here

      -- unload target collection after it is sucessfully st
      p_index := 0;
      p_targetdata.delete;
end storeResults;
```

store

```
open c_sourcedata;
loop

    -- fetch in chunks
    l_sourcedata.delete;
    fetch c_sourcedata bulk collect into l_sourcedata limit

    -- process data
    -- do the mapping between data source and data target
    for i in 1..l_sourcedata.count loop
      -- reset row
      l_targetrow := l_targetrow_empty;

      -- source record to target record
      -- mapping rules
      l_targetrow.id   := l_sourcedata(i).nr;
      l_targetrow.text := l_sourcedata(i).spelling;

      -- store result in collection
      -- the target collection needs to use its own index.
      i_target := i_target + 1;
      l_targetdata(i_target)  := l_targetrow;

    end loop;

    -- store data
    if i_target &amp;amp;gt;= c_target_bulksize then
      storeResults(l_targetdata,i_target);
    end if;

    exit when c_sourcedata%notfound;

end loop;
close c_sourcedata;

-- finally store remaining data
storeResults(l_targetdata,i_target);
```

transform

SQL is always faster than PL/SQL

syntegris

DOGMAS

When and How to Write SQL in Oracle PL/SQL

You should do as much as possible in "pure" SQL

Steven Feuerstein
2014

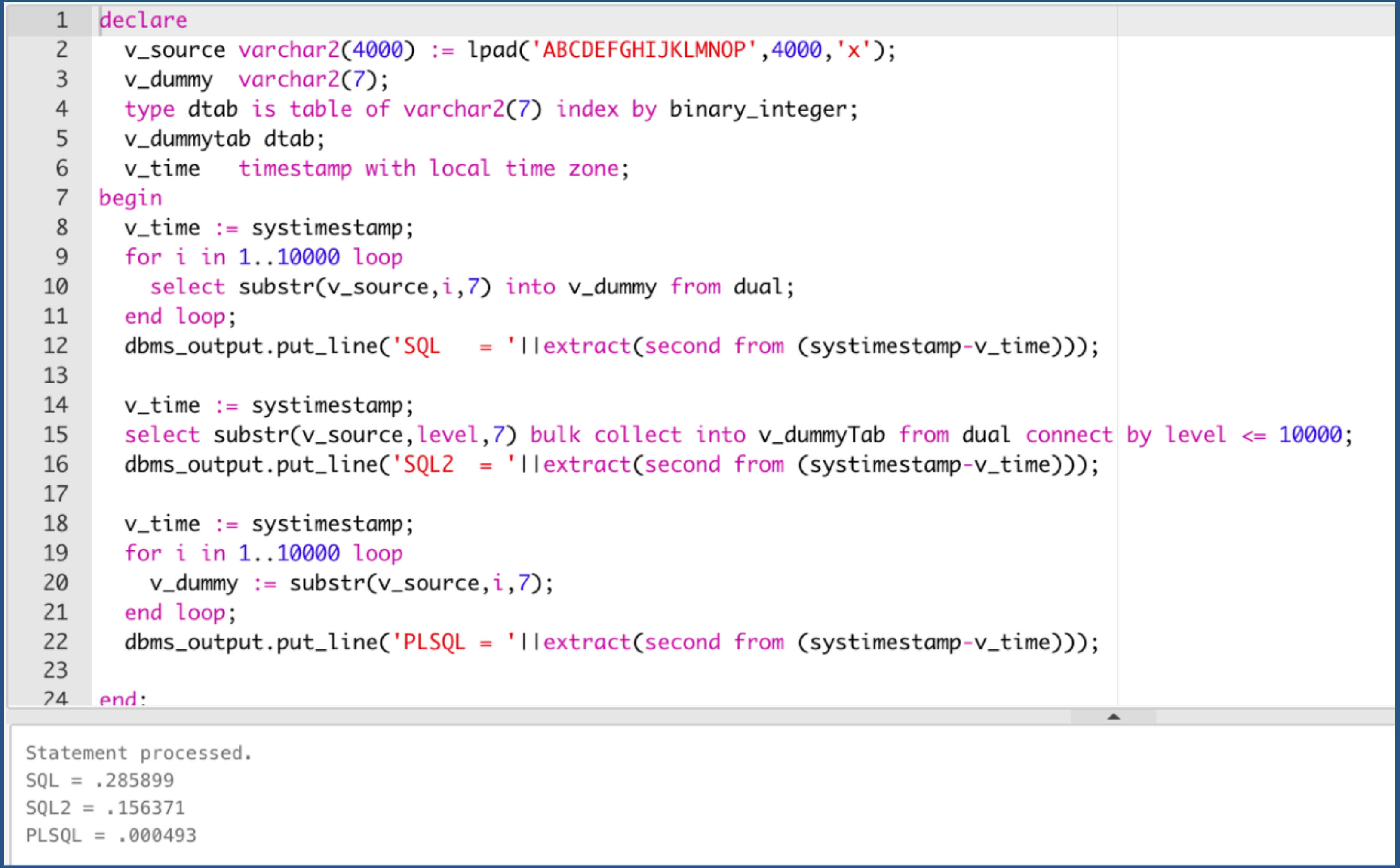
Tom Kyte
Apr 2007

If you can do it in a single SQL statement, by all means do it in a single SQL statement. Do not waste time, energy, and CPU cycles writing procedural code that will run slower than regular SQL.

I have a pretty simple mantra when it comes to developing database software, and I have written this many times over the years:

- You should do it in a single SQL statement if at all possible.
- If you cannot do it in a single SQL statement, do it in PL/SQL.
- If you cannot do it in PL/SQL, try a Java stored procedure.
- If you cannot do it in Java, do it in a C external procedure.
- If you cannot do it in a C external procedure, you might want to seriously think about why it is you need to do it.

SQL is always faster than PLSQL



```
declare
  v_source varchar2(4000) := lpad('ABCDEFGHIJKLMNOP',4000,'x');
  v_dummy  varchar2(7);
  type dtab is table of varchar2(7) index by binary_integer;
  v_dummytab dtab;
  v_time   timestamp with local time zone;
begin
  v_time := systimestamp;
  for i in 1..10000 loop
    select substr(v_source,i,7) into v_dummy from dual;
  end loop;
  dbms_output.put_line('SQL   = '||extract(second from (systimestamp-v_time)));

  v_time := systimestamp;
  select substr(v_source,level,7) bulk collect into v_dummyTab from dual connect by level <= 10000;
  dbms_output.put_line('SQL2  = '||extract(second from (systimestamp-v_time)));

  v_time := systimestamp;
  for i in 1..10000 loop
    v_dummy := substr(v_source,i,7);
  end loop;
  dbms_output.put_line('PLSQL = '||extract(second from (systimestamp-v_time)));

end;
```

```
Statement processed.
SQL = .285899
SQL2 = .156371
PLSQL = .000493
```

SQL is always faster than PLSQL

SQL = 4th GL

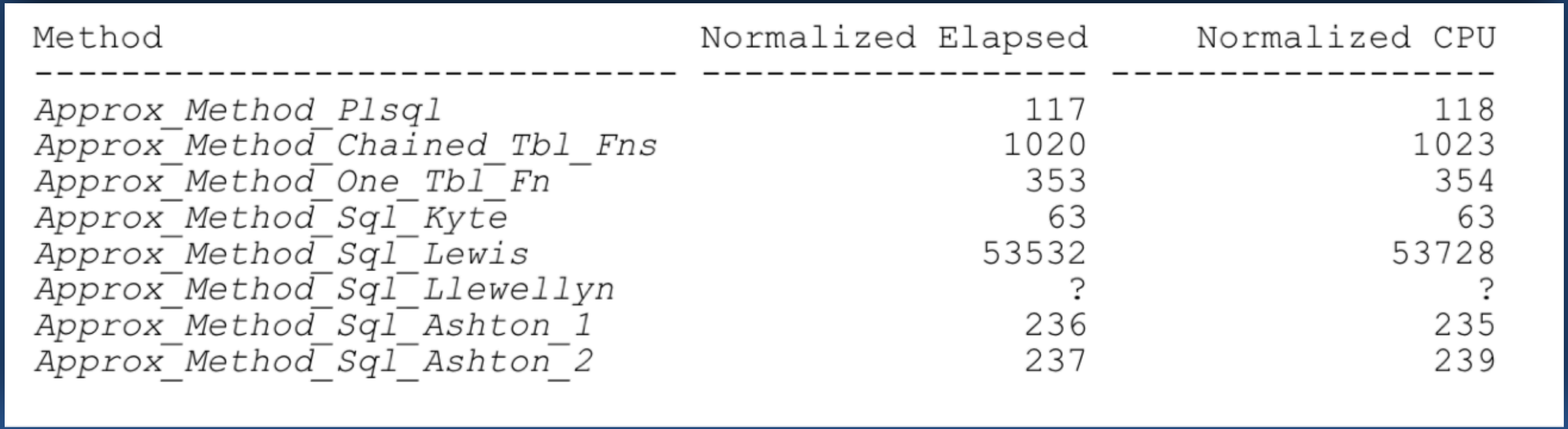
PL/SQL = 3rd GL

Context Switches

Performance vs. Maintainability

SQL is always faster than PLSQL

"How to compute non-overlapping RowID ranges that completely cover a nominated table and that all contain, as nearly as is possible, the same number of rows."



```
Method                          Normalized Elapsed    Normalized CPU
------------------------------ ------------------ ------------------
Approx_Method_Plsql                           117                118
Approx_Method_Chained_Tbl_Fns                1020               1023
Approx_Method_One_Tbl_Fn                      353                354
Approx_Method_Sql_Kyte                         63                 63
Approx_Method_Sql_Lewis                     53532              53728
Approx_Method_Sql_Llewellyn                     ?                  ?
Approx_Method_Sql_Ashton_1                    236                235
Approx_Method_Sql_Ashton_2                    237                239
```

Bryn Llewellyn
Okt 2015

And the winner is...

My choice is *Approx_Method_Plsql*. And that isn't just because I'm Oracle Corporation's product manager for PL/SQL. The fastest pure SQL approach is twice as slow as this. That might not rule it out were it not for the fact that – at least it seems to me – it is rather difficult to understand.

DOGMAS

when OTHERS then null is a BUG

select * is BAD

SQL is always faster than PLSQL

SEQUENCES can not be GAPLESS

CONCLUSION

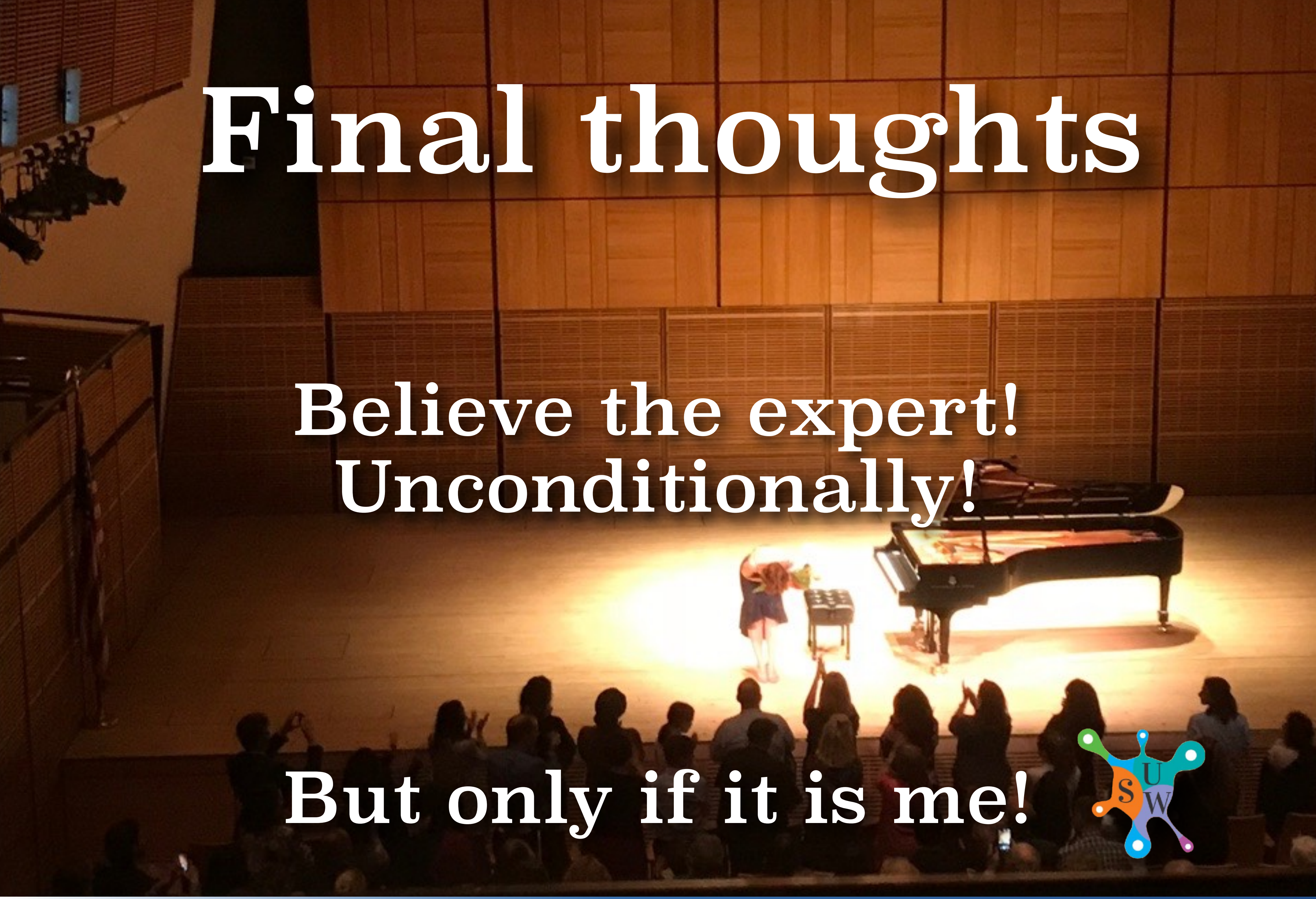
understand WHY

WHO is target

WHEN to use

expert can be WRONG

new features CHANGE

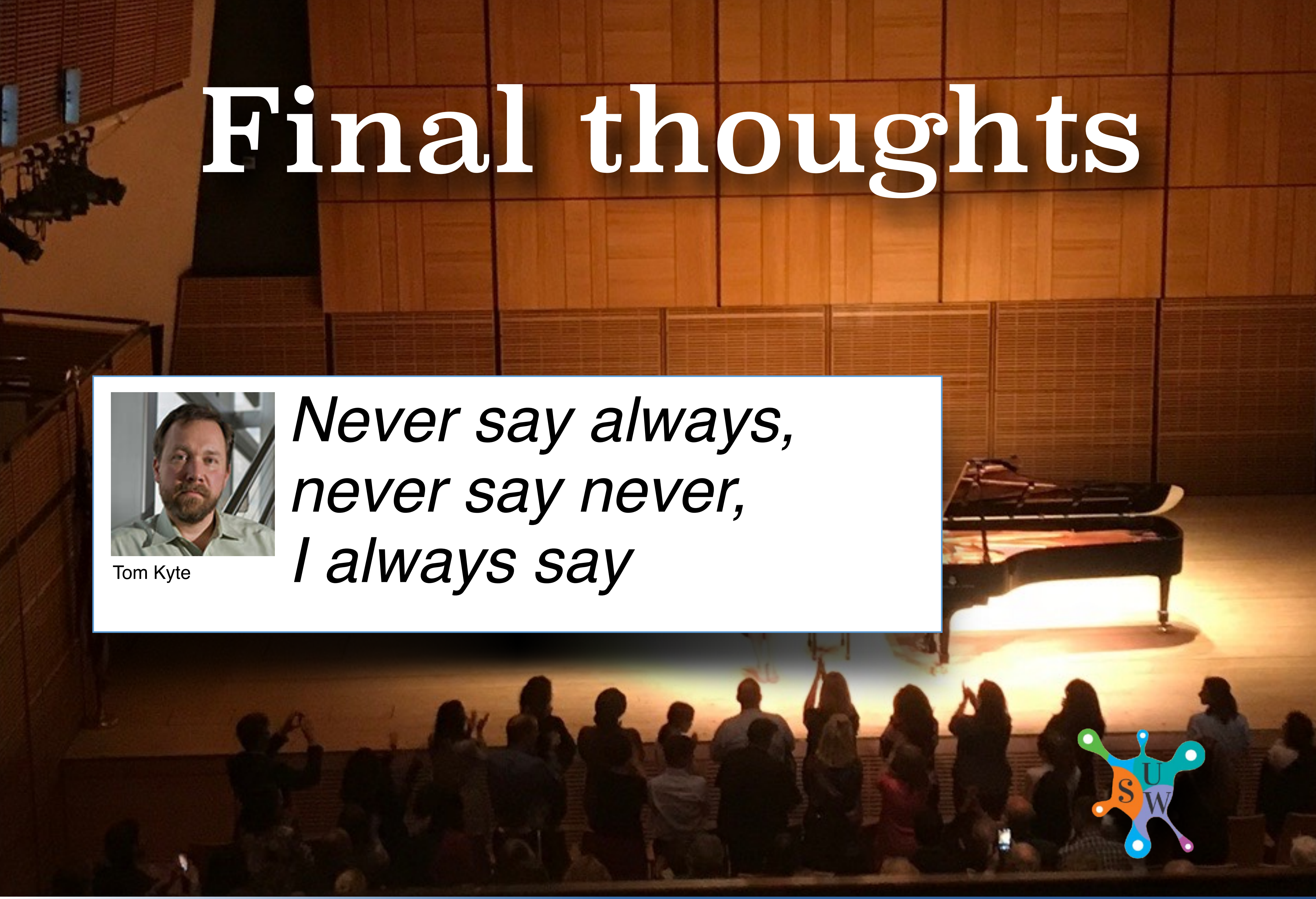


Final thoughts

Believe the expert!
Unconditionally!

But only if it is me!

Final thoughts



Tom Kyte

Never say always,
never say never,
I always say