Expanding the SQL Horizons: PL/SQL User-Defined Functions in the Real World

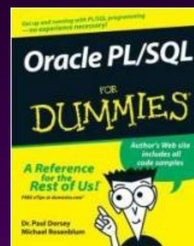
March 12, 2014

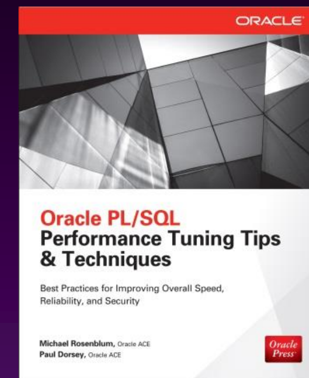
Michael Rosenblum
Dulcian, Inc.
www.dulcian.com

DULCIAN INC.

Who Am I? – “Misha”

- Oracle ACE
- Co-author of 3 books
  - *PL/SQL for Dummies*
  - *Expert PL/SQL Practices*
  - *Oracle PL/SQL Performance Tuning Tips & Techniques* (Rosenblum & Dorsey, Oracle Press, July 2014)





- Won ODTUG 2009 Speaker of the Year
- Known for:
  - SQL and PL/SQL tuning
  - Complex functionality
    - Code generators
    - Repository-based development

DULCIAN INC.

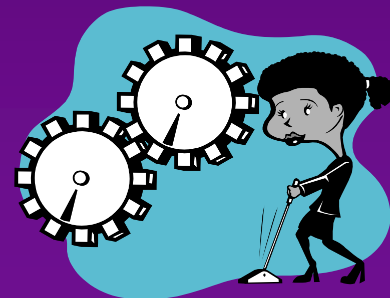
© Tom Kyte

- If you have a thing to do in the database, do it in SQL.
- If you cannot do it in SQL, do it in PL/SQL.
- If you cannot do it in either SQL or PL/SQL, do it in Java.
- If you cannot do it even in Java, do it in C.
- If you cannot do it in C, are you sure that it needs to be done?

DULCIAN INC.

So…

- The rule:
  - Working in the same environment is better than jumping between multiple ones.
- Reasoning:
  - You pay a price for **unnecessary** context switches, especially in terms of CPU cost.

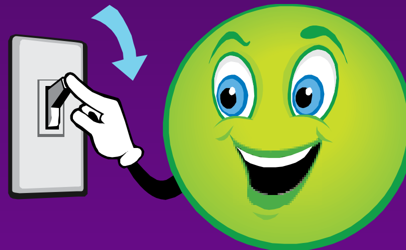


DULCIAN INC.

But…

- The reality is more complicated, because it has lots of dimensions:
  - Performance
  - Maintainability
  - Available expertise, etc.
- If context switches are necessary, it is OK to use multiple environments.

Switching to PL/SQL



Why PL/SQL?

- Flexibility:
  - PL/SQL can utilize Dynamic SQL to build SQL statements on the fly that precisely match the situation
    - … while hardcoded SQL query has to support all possible permutations.
- Functionality:
  - PL/SQL can do whatever you can code
    - … while SQL features are provided to you by Oracle

DULCIAN INC.

Generic Search Problem

- Use Case:
  - Complex generic search
- Common approach:
  - Build a single complex SQL statement that takes all possible filters.
- Common problem #1:
  - Some condition permutations cause major performance issues.
- Common problem #2:
  - Optimizing condition A worsens condition B…
- Common problem #3:
  - Optimizing condition B worsens…

Generic Search Solution

- Proposal:
  - Build SQL statements on the fly.
  - Only some filters (and some tables) can be used at any point in time.
  - Always use bind variables.
  - Use object collection to bring back the result:

```
CREATE TYPE emp_search_ot AS OBJECT
                        (empno_nr NUMBER,
                         empno_dsp VARCHAR2(256),
                         comp_nr   NUMBER);

CREATE TYPE emp_search_nt IS TABLE OF emp_search_ot;
```

DULCIAN INC.

Generic Search (1)

```
create function f_search_tt
                (i_empno number:=null,
                 i_ename_tx varchar2:=null,
                 i_loc_tx varchar2:=null,
                 i_limit_nr number:=50)
return emp_search_nt
is
  v_out_tt emp_search_nt:=emp_search_nt();
  v_from_tx  varchar2(32767):='emp';
  v_where_tx varchar2(32767):=
                           'rownum<=v_limit_nr';
  v_plsql_tx varchar2(32767);
begin
 ...
```

Generic Search (2)

```
if i_empno is not null then
  v_where_tx:=v_where_tx||chr(10)||
                  'and emp.empno=v_empno_nr';
end if;

if i_ename_tx is not null then
  v_where_tx:=v_where_tx||chr(10)||
   'and emp.ename like ''%''||v_ename_tx||''%''';
end if;

if i_loc_tx is not null then
  v_from_tx:=v_from_tx||chr(10)||
        'join dept on  (emp.deptno=dept.deptno)';
  v_where_tx:=v_where_tx||chr(10)||
                  'and dept.loc=v_loc_tx';
end if;
...
```

DULCIAN INC.

Generic Search (3)

```
v_plsql_tx:=
   'declare '||chr(10)||
   'v_limit_nr number:=:1;'||chr(10)||
   'v_empno_nr number:=:2;'||chr(10)||
   'v_ename_tx varchar2(256):=:3;'||chr(10)||
   'v_loc_tx   varchar2(256):=:4;'||chr(10)||
   'begin '||chr(10)||
   'select emp_search_ot('||
      'emp.empno,emp.ename||''(''||emp.job||'')'','||
      'emp.sal+nvl(emp.comm,0))'||chr(10)||
   'bulk collect into :5'||chr(10)||
   'from '||v_from_tx||chr(10)||
   'where '||v_where_tx||';'||chr(10)||
   'end;';
 execute immediate v_plsql_tx using
   i_limit_nr, i_empno, i_ename_tx, i_loc_tx,
   out v_out_tt;
 return v_out_tt;
end;
```

All possible binds are variables

Type constructor

Bulk output

DULCIAN INC.

Generic Search(3)

- Key points:
  - Table DEPT is joined only when needed.
    - ANSI SQL comes in handy when you need to split conditions and joins!
  - Building anonymous PL/SQL block instead of SQL query simplifies passing bind variables.
    - Otherwise you need to cover all possible permutations.
  - EMP_SEARCH_OT constructor must be inside
    - Because the output is an object collection, not a record

DULCIAN INC.

Create your own SQL functionality

- Use Case:
  - Built-in function LISTAGG is limited to 4000 characters
    - … but it would be nice if it could return a CLOB!
- Solution:
  - Build your own aggregate function using Oracle Extensibility Framework
    - … if you can survive ODCI syntax ☺
- Although…
  - Oracle normally does not allow aggregate functions to have multiple parameters (LISTAGG is the exception).
  - Your own aggregate parameter object:

```
CREATE TYPE listAggParam_ot IS OBJECT
                         (value_tx VARCHAR2(4000),
                          separator_tx VARCHAR2(10))
```

ODCI Object Type (1)

```
create or replace type ListAggCLImpl as object (
  v_out_cl clob,
  v_defaultseparator_tx varchar2(10),
  static function ODCIAggregateInitialize
   (sctx in out listaggclimpl) return number,
  member function ODCIAggregateIterate
   (self in out listaggclimpl,
    value_ot in listaggparam_ot) return number,
  member function ODCIAggregateTerminate
   (self in listaggclimpl,
    returnvalue out clob,
    flags in number) return number,
  member function ODCIAggregateMerge
   (self in out listaggclimpl,
    ctx2 in listaggclimpl) return number
  )
```

IN-Type

OUT-Type

ODCI Object Type (2)

- Type elements (mandatory):
  - Type attributes
    - Intermediate data storage
  - method ODCIAggregateInitialize
    - Called once per group to initialize all required settings
  - method ODCIAggregateIterate
    - Called once for every processed value
    - The second parameter’s datatype - input to be aggregated.
  - method ODCIAggregateTerminate
    - Called once at the end of each group
    - The second parameter’s datatype - output of aggregate function.
  - method ODCIAggregateMerge
    - Called if your aggregate function is running in parallel
    - Used to put together the results of different threads.

ODCI Object Type Body (1)

```
create or replace type body listaggclimpl is

static function ODCIAggregateInitialize
 (sctx in out listaggclimpl)
return number is
begin
  sctx :=listaggclimpl(null,',');
  return odciconst.success;
end;

member function ODCIAggregateTerminate
   (self in listaggclimpl,
    returnvalue out clob,
    flags in number)
return number is
begin
  returnvalue := self.v_out_cl;
  return odciconst.success;
end;
```

Default constructor

Return final value

DULCIAN INC.

ODCI Object Type Body (2)

```
member function ODCIAggregateIterate
 (self in out listaggclimpl, value_ot in listaggparam_ot)
return number is
begin
  if self.v_out_cl is null then
    self.v_defaultseparator_tx:=value_ot.separator_tx;
    dbms_lob.createtemporary
          (self.v_out_cl,true,dbms_lob.call);
    dbms_lob.writeappend (self.v_out_cl,
          length(value_ot.value_tx),value_ot.value_tx);
  else
    dbms_lob.writeappend(self.v_out_cl,
        length(value_ot.separator_tx||value_ot.value_tx),
        value_ot.separator_tx||value_ot.value_tx);
  end if;
  return odciconst.success;
end;
```

Initial write

Add separators for consecutive writes

ODCI Object Type Body (3)

```
member function ODCIAggregateMerge
   (self in out listaggclimpl, ctx2 in listaggclimpl)
return number is
begin
  if ctx2.v_out_cl is not null then
    if self.v_out_cl is null then
        self.v_out_cl:=ctx2.v_out_cl;
    else
        dbms_lob.writeappend(self.v_out_cl,
                  length(self.v_defaultseparator_tx),
                  self.v_defaultseparator_tx);
        dbms_lob.append(self.v_out_cl,ctx2.v_out_cl);
    end if;
  end if;
  return odciconst.success;
end;

end;
```

Merge parallel processes in pairs

DULCIAN INC.

ODCI Object Type Explanation

- Key points:
  - Method ODCIAggregateInitialize
    - Has a default constructor to specify the initial values of two attributes of ListAggCLImpl type
  - Method ODCIAggregateIterate
    - Adds new values to the existing temporary storage V_OUT_CL via DBMS_LOB APIs
  - Method ODCIAggregateTerminate
    - Returns the final value of V_OUT_CL as a formal result
  - Method ODCIAggregateMerge
    - Used in case there are parallel executions merging two different V_OUT_CL values into a single output .

Putting everything together

```
CREATE FUNCTION ListAggCL
        (value_ot listAggParam_ot)
RETURN CLOB
PARALLEL_ENABLE
AGGREGATE USING ListAggCLImpl;
```

ListAggCL Usage (1)

```
SQL> SELECT deptno,
  2         ListAggCL(listAggParam_ot(ename,',')) list_cl
  3   FROM emp
  4   GROUP BY deptno;
    DEPTNO LIST_cl
---------- ------------------------------
        10 CLARK,MILLER,KING
        20 SMITH,FORD,ADAMS,SCOTT,JONES
        30 ALLEN,JAMES,TURNER,BLAKE,MARTIN,WARD
```

ListAggCL Usage (2)

Also works as analytical function!

```
SQL> SELECT empno, ename,
  2     ListAggCL(listAggParam_ot(ename,','))
  3       over(partition by deptno
  4             order by ename
  5             rows between unbounded preceding
  6                  and unbounded following
  7              ) list_cl
  8   FROM emp
  9   WHERE job = 'CLERK';
     EMPNO ENAME      LIST_CL
---------- ---------- ----------------------------
      7934 MILLER     MILLER
      7876 ADAMS      ADAMS,SMITH
      7369 SMITH      ADAMS,SMITH
      7900 JAMES      JAMES
```

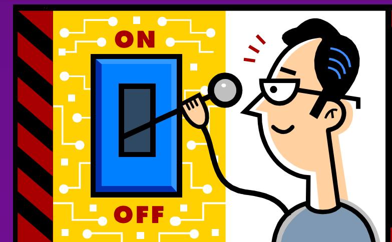
Important

By default, if you have ORDER BY it is "… AND CURRENT ROW"

DULCIAN INC.

Overview

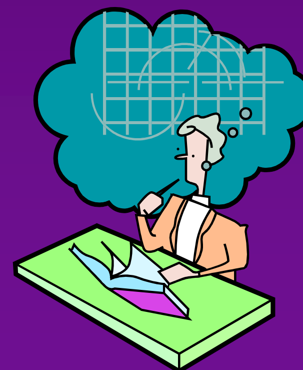
- Switching from SQL to PL/SQL
  - Can:
    - Make your code more flexible and understandable
    - Improve performance in cases when the CBO is confused
    - Expand SQL horizons
  - Cannot:
    - Do it for free
    - There will always be a cost of context switching between language environments.



Calling User-Defined Functions within SQL

THE problem

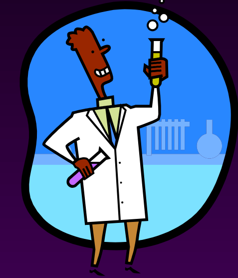
- Question:
  - Do you know how many times your function is being executed within a SQL statement?
- Issue:
  - Few developers know how to properly ask this question
  - … and even fewer know how to correctly interpret the answer!



Reminder!

- SQL statement execution order:
  - 1. JOIN
  - 2. WHERE
  - 3. GROUP BY
  - 4. SELECT (including analytics functions)
  - 5. HAVING
  - 6. ORDER BY

Testing Framework



```
create package counter_pkg is
    v_nr number:=0;
    procedure p_check;
end;

create package body counter_pkg is
  procedure p_check is
  begin
    dbms_output.put_line ('Fired:'||counter_pkg.v_nr);
    counter_pkg.v_nr:=0;
  end;
end;

create function f_change_nr (i_nr number) return
number is
begin
    counter_pkg.v_nr:=counter_pkg.v_nr+1;
    return return i_nr+1;
end;
```

SELECT and WHERE

- Output:

```
SQL> SELECT empno, ename, f_change_nr(empno) change_nr
  2  FROM emp
  3  WHERE f_change_nr(empno) IS NOT NULL
  4  AND deptno = 20;
...
5 rows selected.
SQL> exec counter_pkg.p_check;
Fired: 10
```

Twice the number of returned rows

- Explanation:
  - CBO orders predicates to decrease the total cost
    - DEPNO=20 is applied first to get 5 rows back
    - CBO needs correct info (statistics, indexes, constraints etc.) to make that decision
  - The same functions in SELECT and WHERE clauses are being fired independently.

SELECT and ORDER BY (1)

- Output:

```
SQL> SELECT empno, ename, f_change_nr(empno) change_nr
  2  FROM emp WHERE deptno = 20
  3  ORDER BY 3;
5 rows selected.
SQL> exec counter_pkg.p_check;
Fired: 5
SQL> SELECT empno, change_nr
  2  FROM (SELECT empno, ename, f_change_nr(empno) change
  3          FROM emp WHERE deptno = 20
  4          ORDER BY 3);
5 rows selected.
SQL> exec counter_pkg.p_check;
Fired: 10
```

In-line view causes double-firing

- Problem:
  - Why does the inline view cause a double fire?

DULCIAN INC.™

SELECT and ORDER BY (2)

- Research of 10053 trace:

```
Order-by elimination (OBYE)
***************************
OBYE:     OBYE performed.
OBYE:     OBYE bypassed: no order by to eliminate.

Final query after transformations:*** UNPARSED QUERY IS ***
SELECT "EMP"."EMPNO" "EMPNO","EMP"."ENAME" "ENAME",
       "SCOTT"."F_CHANGE_NR"("EMP"."EMPNO") "CHANGE_NR"
FROM "SCOTT"."EMP" "EMP"
WHERE "EMP"."DEPTNO"=20
ORDER BY "SCOTT"."F_CHANGE_NR"("EMP"."EMPNO")
```

- Explanation:
  - Order-by elimination is fired before the query transformation.
  - It does not find an ORDER BY clause in the root SELECT and stops.
  - After query transformation, ORDER BY suddenly appears.
  - It is not removed now.

SELECT and ORDER BY (3)

- /*+ NO_MERGE */ hint solves the problem:

```
SQL> select empno, change_nr
  2  from (select /*+ no_merge */
  3            empno, ename, f_change_nr(empno) change_nr
  4         from emp where deptno = 20 order by 3);
5 rows selected.
SQL> exec counter_pkg.p_check;
Fired: 5
```

Query rewrite is skipped. → Order-by elimination is applied directly.

- Why bother? Because of views:

```
SQL> create or replace view v_emp as
  2  select empno, ename, f_change_nr(empno) change_nr
  3  from emp where deptno = 20
  4  order by 3;
view created.
SQL> select empno, change_nr from v_emp;
5 rows selected.
SQL> exec counter_pkg.p_check;
Fired: 10
```

Same effect as in-line view

Multi-Table Example

- Join:

```
SQL> SELECT empno, f_change_nr(empno) change_nr, dname
  2  FROM emp,
  3       dept
  4  WHERE emp.deptno(+) = dept.deptno;
     EMPNO  CHANGE_NR DNAME
      7782       7783 ACCOUNTING
      ...
      7654       7655 SALES
                      OPERATIONS
15 rows selected.
SQL> exec counter_pkg.p_check;
Fired: 15
```

No employees!

- Explanation
  - JOIN is being applied first. It results in 15 rows.
  - Function in SELECT is fired as many times as the number of rows.

DULCIAN INC.

Section Summary

- Calling user-defined functions within SQL:
  - Depends on the SQL internal execution order
    - Remember: Joins always happen first!
  - Is impacted by Cost-Based Optimizer decisions:
    - Query transformation
    - Predicate transformations
    - ORDER-BY elimination, etc.

New Cost-Management Features (Oracle 12c)

PRAGMA UDF (1)

- Meaning:
  - PL/SQL function is compiled in the way that is optimized for SQL usage (different memory management).
- Example:

```
CREATE FUNCTION f_change_udf_nr (i_nr NUMBER)
RETURN NUMBER IS
    PRAGMA UDF;
BEGIN
    counter_pkg.v_nr:=counter_pkg.v_nr+1;
    RETURN i_nr+1;
END;
```

PRAGMA UDF (2)

- Much faster in SQL:

```
SQL> SELECT MAX(f_change_nr(object_id))
  2  FROM TEST_TAB;
MAX(F_CHANGE_NR(OBJECT_ID))
---------------------------
                      51485
Elapsed: 00:00:00.48
```

Sample table:
select *
from all_objects
where rownum <=50000

```
SQL> SELECT MAX(f_change_udf_nr(object_id))
  2  FROM TEST_TAB;
MAX(F_CHANGE_UDF_NR(OBJECT_ID))
-------------------------------
                          51485
Elapsed: 00:00:00.06
```

PRAGMA UDF (3)

- Somewhat slower in PL/SQL (1 million iterations):

```
SQL> DECLARE
  2       v_out_nr NUMBER;
  3   BEGIN
  4       FOR i IN 1..1000000 loop
  5           v_out_nr:=f_change_nr(i)+f_change_nr(i+1);
  6       END LOOP;
  7   END;
  8   /
Elapsed: 00:00:01.39
```

```
SQL> DECLARE
  2       v_out_nr NUMBER;
  3   BEGIN
  4       FOR i IN 1..1000000 LOOP
  5          v_out_nr:=f_change_udf_nr(i)+f_change_udf_nr(i+1);
  6       END LOOP;
  7   END;
  8   /
Elapsed: 00:00:01.89
```

PRAGMA UDF Overview

- Looks very promising:
  - Significant performance gains when used in SQL.
  - Minimal performance loss when used in PL/SQL.
- Although…
  - It is a new feature and has not been extensively tested.

DULCIAN INC.

Functions in WITH clause (1)

- Meaning:
  - Functions with the visibility scope of just one SQL query
- Example (the same 50000 row table):

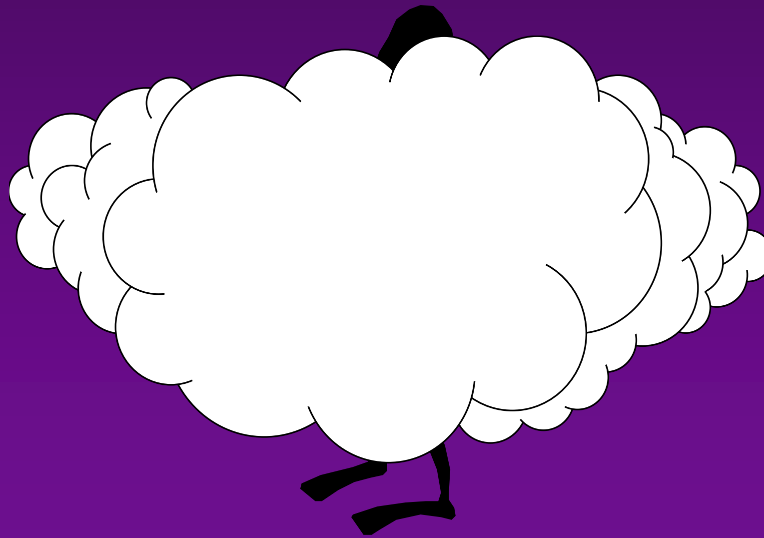
```
SQL> WITH FUNCTION f_changeWith_nr (i_nr number)
  2    RETURN NUMBER IS
  3        BEGIN
  4           RETURN i_nr+1;
  5        END;
  6   SELECT max(f_changeWith_nr(object_id))
  7   FROM test_tab
  8   /
MAX(F_CHANGEWITH_NR(OBJECT_ID))
-------------------------------
                          51485

Elapsed: 00:00:00.07
```

DULCIAN INC.

Functions in WITH clause Overview

- Advantages:
  - Significantly decreases SQL-to-PL/SQL context switches
- Drawbacks:
  - Code fragmentation and duplication
  - PL/SQL limitations – It is not even supported directly (only via Dynamic SQL).
  - SQL limitations - If you want to include it anywhere other than a top-level query, you need a hint /*+ WITH_PLSQL*/
  - Optimization limitations: DETERMINISTIC hint is being ignored.
- Remark:
  - PRAGMA UDF constantly outperforms WITH-clause functions in a number of research papers.
    - Even here it is 0.06 vs 0.07 sec.
- Summary:
  - Usability of this feature is questionable for now.

Query-Level Caching



DULCIAN INC.™

Reasoning

- Question:
  - Why do you need to do something that can be done once and preserved for future use multiple times?
- Answer:
  - You don't!

Same OUT for the same IN

- DETERMINISTIC clause:
  - Definition: If function does the same thing for exactly the same IN, it can be defined as DETERMINISTIC.
  - Impact: Oracle may reuse already calculated values.
  - Warning: Oracle does not check to see whether the function is deterministic or not.
- Example:

```
CREATE FUNCTION f_change_det_tx (i_tx VARCHAR2) RETURN VARCHAR2
DETERMINISTIC IS
BEGIN
    counter_pkg.v_nr:=counter_pkg.v_nr+1;
    RETURN lower(i_tx);
END;
```

Regular counterpart to see the impact

```
CREATE FUNCTION f_change_tx (i_tx VARCHAR2) RETURN VARCHAR2 IS
BEGIN
    counter_pkg.v_nr:=counter_pkg.v_nr+1;
    return lower(i_tx);
END;
```

DULCIAN INC.

DETERMINISTIC (1)

- Basic Output:

```
SQL> select empno, f_change_tx(job) from emp;
SQL> exec counter_pkg.p_check;
Fired:14
SQL> select empno, f_change_det_tx(job) from emp;
SQL> exec counter_pkg.p_check;
Fired:5
```

- Problem:
  - DETERMINISTIC is a hint, not a directive and can be ignored.

- Alternate test - increase the scope (50000 rows table!):

```
alter table test_tab add (obj3_tx varchar2(3),
                          obj1_tx varchar2(1));
update test_tab set
    obj3_tx  = upper(substr(object_name,-3)), -- 3442 distinct
    obj1_tx  = upper(substr(object_name,1,1)); -- 26 distinct

create type stringlist_tt is table of varchar2(256);
```

DETERMINISTIC (2)

```
SQL> declare
  2    v_obj_tt stringlist_tt;
  3    v_count_nr number;
  4    cursor c_rec is
  5       select f_change_det_tx(obj1_tx) obj_tx from test_tab;
  6  begin
  7    open c_rec;
  8    for i in 1..5 loop
  9        fetch c_rec bulk collect into v_obj_tt limit 100;
 10        select count(distinct column_value) into v_count_nr
 11        from table(cast (v_obj_tt as stringlist_tt));
 12        counter_pkg.p_check;
 13        dbms_output.put_line('-real count:'||v_count_nr);
 14    end loop;
 15    close c_rec;
 16  end;
Fired:17 - real count:14
Fired:22 - real count:14
Fired:26 - real count:16
Fired:25 - real count:14
Fired:17 - real count:15
```

DULCIAN INC.

DETERMINISTIC (3)

- Problem:
  - Number of function calls is lower than 200, but not as low and the number of distinct values.
- Explanation:
  - The scope of the DETERMINISTIC clause is one fetch operation (not the whole SQL statement!)
  - Internally, Oracle has a hash table that preserves IN/OUT pairs.
  - By default, this hash table has limited size (65536 bytes) and fills up very quickly.
  - … And you can adjust this size ☺

DETERMINISTIC (4)

- Example (increase hash table size 4 times):

```
SQL> ALTER SESSION
 2       SET "_query_execution_cache_max_size"=262144;
SQL> ... rerun the last example …
Fired:17 - real count:14
Fired:16 - real count:14
Fired:18 - real count:16
Fired:24 - real count:14
Fired:17 - real count:15
```

Originally was:
17
22
26
25
17

- Remarks:
  - It is underscore-parameter, so no Oracle support on this one!
  - Hash collisions between multiple values of IN-parameters can occur
    - That's why you rarely get the number of calls matching the number of distinct values.

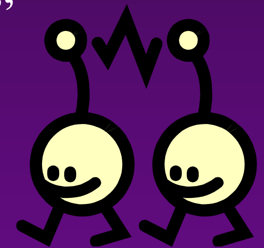
DULCIAN INC.

DETERMINISTIC Overview

- Good:
  - Can improve performance
  - Very light in terms of system impact and management
- Bad:
  - It is a HINT – so you cannot rely on it 100%
  - Adding it inappropriately can cause data problems.
- Ugly:
  - You can adjust internal hash table size, but only at your own risk.
  - Data order may impact the level of optimization.

Side Effect of SELECT from DUAL

- Definitions:
  - Scalar sub-query returns a single column of a single row (or from the empty rowset)
  - Scalar sub-query caching is an Oracle internal mechanism to preserve results of such queries while processing more complex ones.
    - Implemented as in-memory hash table
    - Cache size is defined by "_query_execution_cache_max_size"
    - Cache is preserved for the duration of the query.
    - Last value is preserved even if hash table is full.



- Example:

```
SQL> SELECT empno, f_change_tx(job) FROM emp;
SQL> exec counter_pkg.p_check;
Fired:14
SQL> SELECT empno, (SELECT f_change_tx(job) FROM dual)
  2    FROM emp;
SQL> exec counter_pkg.p_check;
Fired: 5
```

Only 5 distinct jobs

DULCIAN INC.™

Cache Duration

```
sql> declare
  2     v_obj_tt stringlist_tt;
  3     v_count_nr number;
  4     cursor c_rec is select
  5       (select f_change_tx(obj1_tx) from dual) obj_tx from test_tab;
  6   begin
  7     open c_rec;
  8     for i in 1..5 loop
  9       fetch c_rec bulk collect into v_obj_tt limit 100;
 10       select count(distinct column_value)
 11       into v_count_nr from table(cast (v_obj_tt as stringlist_tt));
 12       counter_pkg.p_check;
 13       dbms_output.put_line('-real count:'||v_count_nr);
 14     end loop;
 15     close c_rec;
 16   end;
Fired:17 - real count:14
Fired:16 - real count:14
Fired:10 - real count:16
Fired:18 - real count:14
Fired:5  - real count:15
```

Deterministic:
17
22
26
25
17

Count is less than number of distinct values: Cache duration is query.

DULCIAN INC.

Scalar Sub-Query Overview

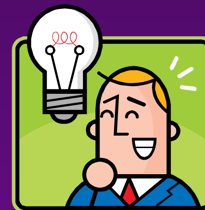
- Good:
  - Can improve performance
  - No changes to PL/SQL code!
  - More reliable and efficient than DETERMINISTIC
- Bad:
  - Can be impacted by query transformation
  - Confusing syntax
- Ugly:
  - You can adjust internal hash table size, but only at your own risk.
  - Data order may impact the level of optimization.

DULCIAN INC.

PL/SQL Result Cache

The Idea

- PL/SQL Function Result Cache
  - Database-level cache (cross-session)
  - Stored in SGA
  - Automatic cache monitoring and invalidation (starting with Oracle 11g R2)
- Example:



```
create function f_getdept_dsp (i_deptno number)
return varchar2 result_cache is
    v_out_tx varchar2(256);
begin
    if i_deptno is null then return null; end if;
    select initcap(dname) into v_out_tx
    from dept where deptno=i_deptno;
    counter_pkg.v_nr:=counter_pkg.v_nr+1;
    return v_out_tx;
end;
```

Result Cache Basics

```
SQL> SELECT empno, f_getDept_dsp(deptno) dept_dsp
  2   FROM emp;
     EMPNO   DEPT_DSP
---------- ------------------------
      7369   Research
       ...
14 rows selected.
SQL> exec counter_pkg.p_check;
Fired:3

SQL> SELECT empno, f_getDept_dsp(deptno) dept_dsp
   2 FROM emp;
     EMPNO DEPT_DSP
---------- ----------
      7369 Research
       ...
14 rows selected.
SQL> exec counter_pkg.p_check;
Fired:0
```

No calls at all!

Result Cache Stats

```
SQL> SELECT * FROM v$result_cache_statistics;
   ID NAME                              VALUE
----- ------------------------------ ----------
    1 Block Size (Bytes)             1024
    2 Block Count Maximum            15360
    3 Block Count Current            32
    4 Result Size Maximum (Blocks)   768
    5 Create Count Success           3
    6 Create Count Failure           0
    7 Find Count                     25
    8 Invalidation Count             0
    9 Delete Count Invalid           0
   10 Delete Count Valid             0
   11 Hash Chain Length              1
   12 Find Copy Count                25
```

3 distinct INs

Result Cache Dependencies

```
SQL> SELECT rco.id,  rco.name, ao.object_name object_name
  2  FROM   v$result_cache_objects     rco,
  3         v$result_cache_dependency rcd,
  4         all_objects               ao
  5  WHERE  rco.id = rcd.result_id
  6  AND    rcd.object_no = ao.object_id
  7  order by 1;

ID NAME                                      OBJECT_NAME
-- ----------------------------------------- -------------
 1 "SCOTT"."F_GETDEPT_DSP"::8."F_ GETDEPT_DSP"  DEPT
 1 "SCOTT"."F_GETDEPT_DSP"::8.”F_ GETDEPT_DSP"  F_GETDEPT_DSP
 3 "SCOTT"."F_GETDEPT_DSP"::8."F_ GETDEPT_DSP"  DEPT
 3 "SCOTT"."F_GETDEPT_DSP"::8.”F_ GETDEPT_DSP"  F_GETDEPT_DSP
 4 "SCOTT"."F_GETDEPT_DSP"::8."F_ GETDEPT_DSP"  DEPT
 4 "SCOTT"."F_GETDEPT_DSP"::8.”F_ GETDEPT_DSP"  F_GETDEPT_DSP
```

Result Cache Cost

- It is not free:

```
SQL> exec runstats_pkg.rs_start
SQL> SELECT MAX(f_change_cache_tx(obj1_tx)) FROM test_tab;
SQL> exec counter_pkg.p_check;
Fired:26

SQL> exec runstats_pkg.rs_middle
SQL> SELECT MAX(f_change_det_tx(obj1_tx)) FROM test_tab;
SQL> exec counter_pkg.p_check;
Fired:7627

SQL> exec runstats_pkg.rs_stop
Run1 ran in 65 cpu hsecs
Run2 ran in 16 cpu hsecs
Name                                  Run1        Run2        Diff
STAT...CPU used by this session         64          16         -48
LATCH.Result Cache: RC Latch        50,079           0     -50,079
Run1 latches total versus runs -- difference and pct
Run1          Run2          Diff        Pct
52,388        2,057       -50,331   2,546.82%
```

DETERMINISTIC is faster because of latching.

DULCIAN INC.

PL/SQL Result Cache Overview

- Good:
  - Database-level caching
  - Automatic dependency tracking
  - Extended statistics
  - Very informative data dictionary views
- Bad:
  - Significant memory utilization
- Ugly:
  - Noticeable CPU impact when retrieving values from cache (because of latching)
  - Too easy to add without real understanding

That's all Folks!

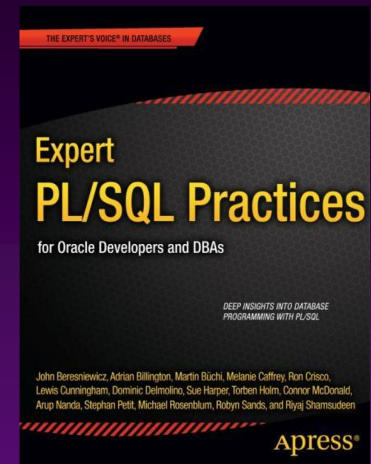
DULCIAN INC.

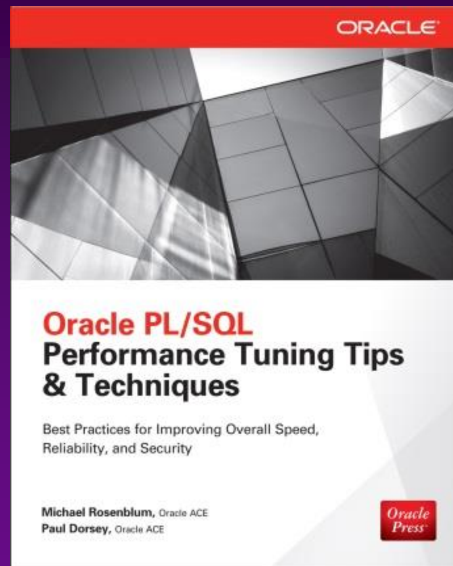
Summary

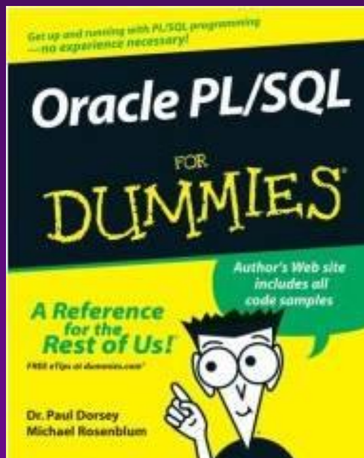
- PL/SQL can significantly extend SQL functionality.
- You <u>must</u> worry about the total number of user-defined function calls.
- PRAGMA UDF can decrease the cost of function calls within SQL.
- Different caching techniques can improve the overall system performance.

Contact Information

- Michael Rosenblum – mrosenblum@dulcian.com
- Blog – wonderingmisha.blogspot.com
- Website – www.dulcian.com







Coming in July 2014

Oracle PL/SQL Performance Tuning Tips & Techniques