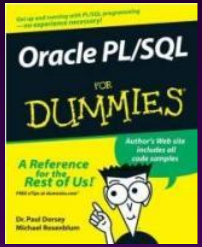
Calling SQL from PL/SQL: The Right Way

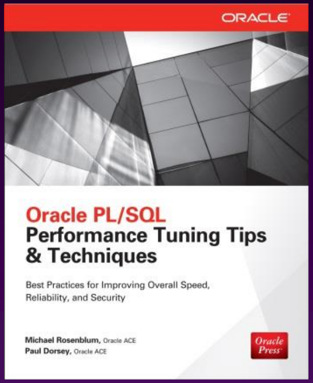
Michael Rosenblum
Dulcian, Inc.

www.dulcian.com

Who Am I? – “Misha”

- Oracle ACE
- Co-author of 3 books
  - *PL/SQL for Dummies*
  - *Expert PL/SQL Practices*
  - *PL/SQL Performance Tuning (July 2014)*




ORACLE
Oracle PL/SQL
Performance Tuning Tips & Techniques
Best Practices for Improving Overall Speed, Reliability, and Security
Michael Rosenblum, Oracle ACE
Paul Dorsey, Oracle ACE
Oracle Press

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

Groundwork

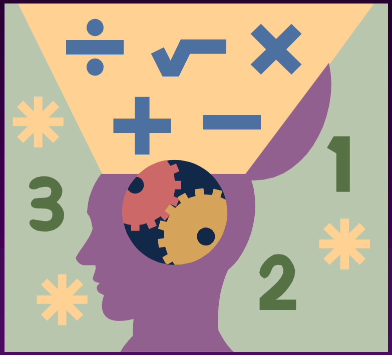
Oracle Optimization

- It is all about CURSORs!
- Using cursors does not imply only row-by-row processing because…
  - Cursors point to SETs
  - Even internally Oracle is using bulk optimization
    - Pre-fetching 100 rows at a time
    - Started in Oracle 10g

Proof (1)

```
create table test_tab as
select *
from all_objects
where rownum <= 50000;

declare
   v_nr number;
begin
  dbms_monitor.session_trace_enable
                      (waits=>true, binds=>true);
   for c in (select * from test_tab
                 where rownum < 1000)
   loop
         v_nr:=c.object_id;
   end loop;
   dbms_monitor.session_trace_disable;
end;
```



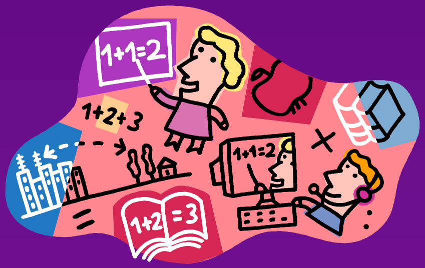
Proof (2)

```
-- TKPROF output

SQL ID: dyxt87m2np50t Plan Hash: 1165077207
SELECT * FROM TEST_TAB WHERE ROWNUM < 1000


call     count        rows
------- ------  ----------
Parse        1           0
Execute      1           0
Fetch       10         999
------- ------  ----------
total       12         999
```

Last fetch returns less than 100 rows



Proof (3)

```
declare
   v_nr number;
begin
   dbms_monitor.session_trace_enable(waits=>true, binds=>true);
   for c in (select * from test_tab where rownum < 1001) loop
       v_nr:=c.object_id;
   end loop;
   dbms_monitor.session_trace_disable;
end;

-- TKPROF output
SQL ID: 544c85gf7tn8f Plan Hash: 1165077207
SELECT * FROM TEST_TAB WHERE ROWNUM < 1001

call      count         rows
------- ------  ----------
Parse         1            0
Execute       1            0
Fetch        11         1000
------- ------  ----------
total        12         1000
```

Last fetch returns 100 rows, so one extra is needed.

So…

If Oracle is using sets internally, you should start using them too!

Loading Sets from SQL to PL/SQL

What kind of Sets?

- Oracle collection datatypes:
  - Nested tables
    - Also called object collections
    - In both SQL and PL/SQL
  - VARRAYs
    - In both SQL and PL/SQL
  - Associative arrays
    - Also called PL/SQL tables or INDEX-BY Tables
    - Two variations: INDEX BY BINARY_INTEGER or INDEX BY VARCHAR2
    - PL/SQL-only

Usability

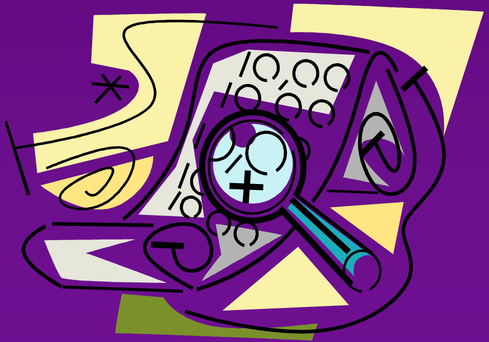
- Associative arrays are useful when:
  - You work only within PL/SQL.
  - You need the index to be a text instead of a number.
- Nested tables are useful when:
  - You need to use collections both in SQL and PL/SQL.
- VARRAYS are useful….
  - Sorry, never needed in the last 15 years…

Research

- Task:
  - Data needs to be retrieved from a remote location via DBLink.
  - Each row has to be processed locally.
  - Source table contains 50,000 rows.
- Problem:
  - Analyze different ways of achieving the goal.
  - Create best practices.

Use Case #1

- Extreme options:
  - Row-by-row processing
  - BULK COLLECT everything in the local object collection beforehand
- Limited scope:
  - Only one column from the source table is touched



Use Case #1 – BULK

```
SQL> connect scott/TIGER@localDB;
sql> declare
  2       type number_tt is table of number;
  3       v_tt number_tt;
  4       v_nr number;
  5   begin
  6       select object_id
  7       bulk collect into v_tt
  8       from test_tab@remotedb;
  9       for i in v_tt.first..v_tt.last loop
 10           v_nr:=v_tt(i);
 11       end loop;
 12   end;
 13   /
Elapsed: 00:00:00.09

SQL> select name, value from stats where name in
  2     ('STAT...session pga memory max',
  3        'STAT...SQL*Net roundtrips to/from dblink');
NAME                                         VALUE
----------------------------------------- -------
STAT...session pga memory max              3330400
STAT...SQL*Net roundtrips to/from dblink        10
```

Use Case #1 – RowByRow

```
SQL> connect scott/TIGER@localDB;
sql> declare
  2     v_nr number;
  3  begin
  4     for c in  (select object_id
  5                from test_tab@remotedb)
  6      loop
  7         v_nr :=c.object_id;
  8     end loop;
  9  end;
 10  /
Elapsed: 00:00:00.42

SQL> select name, value from stats where name in
  2       ('STAT...session pga memory max',
  3        'STAT...SQL*Net roundtrips to/from dblink');
NAME                                            VALUE
----------------------------------------- -------
STAT...session pga memory max             2543968
STAT...SQL*Net roundtrips to/from dblink      510
```

Use Case #1 - Analysis

- Results:

	Bulk	Row By Row
Processing Time	0.09	0.42
PGA memory max	3’330’400	2’543’968
SQL*Net roundtrips to/from dblink	10	510

- Summary:
  - BULK COLLECT is faster and less network-intensive
  - … but it uses more memory – even for a single column!
- Conclusion:
  - More tests are needed!

Use Case #2

- Scope change:
  - Get all columns from the source table (15 total)

```
Name                           Data Type
------------------------------ ------------------------------
OWNER                          VARCHAR2(30 BYTE)     NOT NULL
OBJECT_NAME                    VARCHAR2(30 BYTE)     NOT NULL
SUBOBJECT_NAME                 VARCHAR2(30 BYTE)
OBJECT_ID                      NUMBER                NOT NULL
DATA_OBJECT_ID                 NUMBER
OBJECT_TYPE                    VARCHAR2(19 BYTE)
CREATED                        DATE                  NOT NULL
LAST_DDL_TIME                  DATE                  NOT NULL
TIMESTAMP                      VARCHAR2(19 BYTE)
STATUS                         VARCHAR2(7 BYTE)
TEMPORARY                      VARCHAR2(1 BYTE)
GENERATED                      VARCHAR2(1 BYTE)
SECONDARY                      VARCHAR2(1 BYTE)
NAMESPACE                      NUMBER                NOT NULL
EDITION_NAME                   VARCHAR2(30 BYTE)
```

Use Case #2 – BULK

```
sql> connect scott/tiger@localdb;
sql> declare
  2    type table_tt is table of test_tab@remotedb%rowtype;
  3    v_tt table_tt;
  4    v_nr number;
  5  begin
  6    select *
  7    bulk collect into v_tt
  8    from test_tab@remotedb;
  9    for i in v_tt.first..v_tt.last loop
 10        v_nr:=v_tt(i);
 11    end loop;
 12  end;
 13  /
Elapsed: 00:00:00.51

SQL> select name, value from stats where name in
  2    ('STAT...session pga memory max',
  3       'STAT...SQL*Net roundtrips to/from dblink');
NAME                                        VALUE
----------------------------------------- -------
STAT...session pga memory max             34656608
STAT...SQL*Net roundtrips to/from dblink        10
```

Use Case #2 – RowByRow

```
SQL> connect scott/TIGER@localDB;
sql> declare
  2       v_nr number;
  3   begin
  4       for c in  (select * from test_tab@remotedb) loop
  5           v_nr :=c.object_id;
  6       end loop;
  7   end;
  8   /
Elapsed: 00:00:00.77

SQL> select name, value from stats where name in
  2      ('STAT...session pga memory max',
  3       'STAT...SQL*Net roundtrips to/from dblink');
NAME                                        VALUE
---------------------------------------- -------
STAT...session pga memory max             2609504
STAT...SQL*Net roundtrips to/from dblink      510
```

Use Case #2 - Analysis

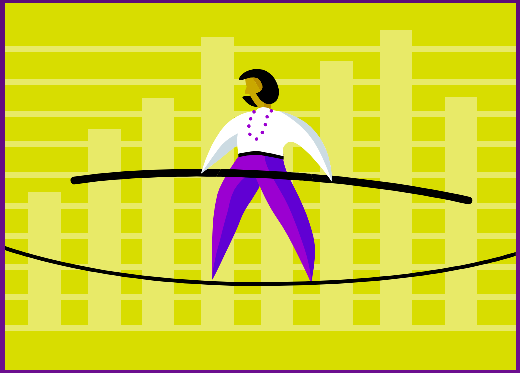
- Results:

	Bulk	Row By Row
Processing Time	0.51	0.77
PGA memory max	34’656’608	2’609’504
SQL*Net roundtrips to/from dblink	10	510

- Summary:
  - BULK COLLECT is still faster
  - … but memory usage is wa-a-a-ay up!
- Conclusion:
  - “Bulk everything” may cause major problems if your system is memory-bound!
  - It may cause the database to slow down.

Use Case #3

- Walking the line:
  - FETCH … BULK COLLECT LIMIT <N> decreases the memory workload, while still using bulk operations.
  - It does not make sense to test limits less than 100 because that is Oracle’s internal pre-fetch size.



Use Case #3 – BULK LIMIT

Limit can variable

```
sql> declare
  2       type collection_tt is table of
  3                test_tab@remotedb%rowtype;
  4       v_tt collection_tt;
  5       v_nr number;
  6       v_cur sys_refcursor;
  7       v_limit_nr binary_integer:=5000;
  8   begin
  9       open v_cur for select * from test_tab@remotedb;
 10       loop
 11           fetch v_cur bulk collect into v_tt
 12               limit v_limit_nr;
 13           exit when v_tt.count()=0;
 14           for i in v_tt.first..v_tt.last loop
 15               v_nr:=v_tt(i).object_id;
 16           end loop;
 17           exit when v_tt.count<v_limit_nr;
 18       end loop;
 19       close v_cur;
 20   end;
 21   /
```

Use Case #3 - Analysis

- Results:

Limit size	Time	Max PGA	Roundtrips
100	0.78	2’543’968	510
250	0.58	2’675’040	210
500	0.49	2’806’112	110
1000	0.44	3’133’792	60
5000	0.40	4’247’904	20
10000	0.41	7’590’240	15
20000	0.43	14’340’448	12

- Summary:
  - With the increase of bulk limit processing, time stops dropping because memory management becomes costly!
  - This point is different for different hardware/software
- Conclusion:
  - Run your own tests and find the most efficient bulk limit

Pagination vs. Continuous Fetch

Row-Limiting Clause (1)

- New feature in Oracle 12c

```
SELECT …
FROM …
WHERE …
OFFSET <N> ROWS
FETCH NEXT <rowcount ROWS|percent PERCENT>
   <ONLY|WITH TIES>
```

- Question: Can it be an alternative to continuous fetch?

Row-limiting clause (2)

```
SQL> exec runstats_pkg.rs_start;

     <... run BULK COLLECT LIMIT 5000 ...>

sql> exec runstats_pkg.rs_middle;
sql> declare
         type table_tt is table of test_tab%rowtype;
         v_tt table_tt;
         v_nr number;
         v_limit_nr constant number:=5000;
         v_counter_nr number:=0;
     begin
         loop
             select *
             bulk collect into v_tt
             from test_tab
             offset v_counter_nr*v_limit_nr rows
             fetch next 5000 rows only;
             exit when v_tt.count()=0;
             for i in v_tt.first..v_tt.last loop
                 v_nr:=v_tt(i).object_id;
             end loop;
             exit when v_tt.count<v_limit_nr;
            v_counter_nr:=v_counter_nr+1;
         end loop;
     end;
     /
```

No DBLink - less moving parts!

Limitation/bug:
has to be hardcoded for now

Row-Limiting Clause Analysis

- Results:

```
SQL> exec runstats_pkg.rs_stop;
Run1 ran in 33 cpu hsecs
Run2 ran in 78 cpu hsecs
Name                                           Run1        Run2
STAT...consistent gets                          900       5,360
STAT...logical read bytes from cache      7,331,840  44,269,568
```

- Summary:
  - Row-limiting clause cannot substitute continuous fetch
  - it causes tables to be re-read multiple times.

Proof of Multiple Reads

- Original query:

```
SELECT *
FROM test_tab
OFFSET 5000 ROWS
FETCH NEXT 5000 ROWS ONLY
```

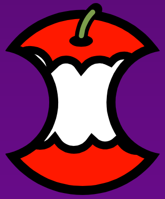
- Section “Unparsed Query” of 10053 trace:

```
SELECT   …
FROM   (SELECT   "TEST_TAB".*,
             ROW_NUMBER () OVER (ORDER BY NULL) "rowlimit_$$_rownumber"
        FROM   "HR"."TEST_TAB" "TEST_TAB") "from$_subquery$_002"
WHERE   "from$_subquery$_002"."rowlimit_$$_rownumber" <=
         CASE WHEN (5000 >= 0) THEN FLOOR (TO_NUMBER (5000)) ELSE 0 END
             + 5000
         AND "from$_subquery$_002"."rowlimit_$$_rownumber" > 5000
```

Merging Sets in PL/SQL

Core Use Case

- The story:
  - Generic “Attention” folder with a lot of conditions is supported by a single view.
  - Each set of conditions is represented by a SQL query.
  - Results are merged using UNION ALL.
- Problem:
  - View became unmaintainable.



- Solution:
  - PL/SQL function that returns object collection.

Code Sample

```
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;

create function f_attention_ot (i_empno number) return emp_search_nt is
   v_emp_rec emp%rowtype;
    v_sub_nt   emp_search_nt;
    v_comm_nt  emp_search_nt;
    v_out_nt   emp_search_nt;
begin
    -- load information about the logged user
    select * into v_emp_rec from emp where empno=i_empno;
    -- get subordinates
    if v_emp_rec.job = 'manager' then -- directly reporting
   ... query 1 ... into v_sub_nt ...
    elsif v_emp_rec.job = 'president' then -- get everybody except himself
   ... query 2 ... into v_sub_nt ...
    end if;

    -- check all people with commissions from other departments
    if v_emp_rec.job in ('manager','analyst') then
   ... query 1 ... into v_comm_nt ...
    end if;

    -- merge two collection together
    v_out_nt:=v_sub_nt multiset union distinct v_comm_nt;

    return v_out_nt;
end;
```

Cost of MULTISET(1)

- Question:
  - How much overhead is created by doing MULTISET operations instead of pure SQL?
- Answer:
  - Interesting to know!

- Test setup:
  - Read large number of rows (~36,000 total) out of two similar tables (50,000 rows each) and put them together using all available mechanisms

Cost of MULTISET – Setup(1)

```
create table test_tab2 as select * from test_tab;

create type test_tab_ot as object
        (owner_tx varchar2(30), name_tx varchar2(30),
         object_id number, type_tx varchar2(30));
create type test_tab_nt is table of test_tab_ot;

create function f_seachtesttab_tt (i_type_tx varchar2)
return test_tab_nt is
    v_out_tt test_tab_nt;
begin
    select test_tab_ot(owner, object_name, object_id, object_type)
    bulk collect into v_out_tt
    from test_tab
    where object_type = i_type_tx;
    return v_out_tt;
end;

create function f_seachtesttab2_tt (i_type_tx varchar2)
return test_tab_nt is
    v_out_tt test_tab_nt;
begin
    select test_tab_ot(owner, object_name, object_id, object_type)
    bulk collect into v_out_tt
    from test_tab2
    where object_type = i_type_tx;
    return v_out_tt;
end;
```

Cost of MULTISET – Setup(2)

```
create function f_seachtestunion_tt
   (i_type1_tx varchar2, i_type2_tx varchar2)
return test_tab_nt is
    v_type1_tt test_tab_nt;
    v_type2_tt test_tab_nt;
    v_out_tt test_tab_nt;
begin
    select test_tab_ot(owner, object_name, object_id, object_type)
    bulk collect into v_type1_tt
    from test_tab
    where object_type = i_type1_tx;

    select test_tab_ot(owner, object_name, object_id, object_type)
    bulk collect into v_type2_tt
    from test_tab2
    where object_type = i_type2_tx;

    v_out_tt:= v_type1_tt multiset union all v_type2_tt;

    return v_out_tt;
end;
```

Cost of MULTISET – Run

```
-- Run1: Union of SQL queries
SQL> select max(object_id), min(object_id), count(*)
  2  from (select * from test_tab
  3           where object_type = 'SYNONYM'
  4           union all
  5           select * from test_tab2
  6           where object_type = 'JAVA CLASS');

-- Run 2: UNION of collections
SQL> select max(object_id), min(object_id), count(*)
  2  from (select *
  3           from table(f_seachTestTab_tt('SYNONYM'))
  4           union all
  5           select *
  6           from table(f_seachTestTab2_tt('JAVA CLASS')));

-- Run 3: UNION inside PL/SQL
SQL> select max(object_id), min(object_id), count(*)
  2  from table
  3        (f_seachTestUnion_tt('SYNONYM','JAVA CLASS'));|
```

Cost of MULTISET - Analysis

- Results:

	Time	Max PGA
Union of SQL	0.05	1’564’840
Union of collections	0.55	14’409’896
MULTISET union	0.68	40’231’080

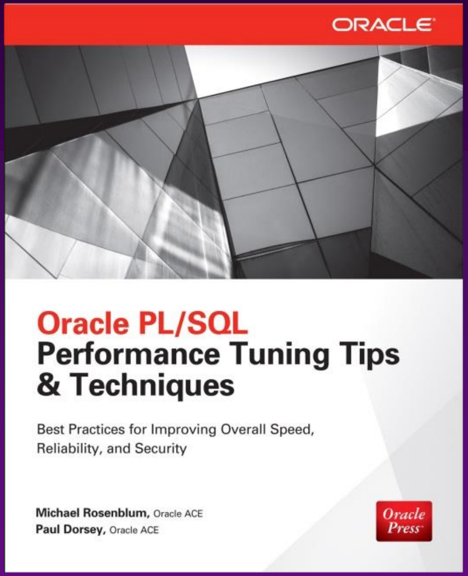
- Summary:
  - Collection operations cause significant memory overhead and slowdown.
- Conclusions:
  - With a small data set, you can trade some performance for maintainability.
  - With large data sets, you should stay with SQL as long as you can and switch to PL/SQL only when you don’t have any other choice.

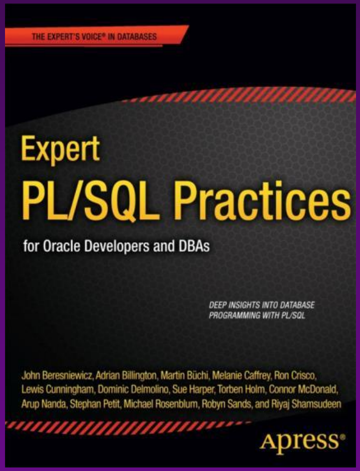
Summary

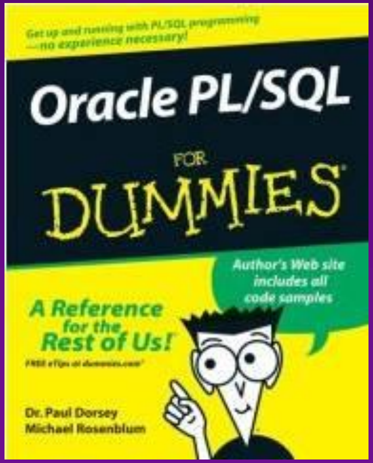
- You must think in sets when integrating SQL into PL/SQL.
  - Otherwise, your solutions would never scale.
- Always keep the overhead cost of object collection memory management in mind .
- The most effective bulk size depends upon your database configuration
  - … but is usually between 100 and 10,000

Contact Information

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


ORACLE
Oracle PL/SQL
Performance Tuning Tips
& Techniques
Best Practices for Improving Overall Speed, Reliability, and Security
Michael Rosenblum, Oracle ACE
Paul Dorsey, Oracle ACE
Oracle Press


THE EXPERT'S VOICE® IN DATABASES
Expert
PL/SQL Practices
for Oracle Developers and DBAs
DEEP INSIGHTS INTO DATABASE PROGRAMMING WITH PL/SQL
John Beresniewicz, Adrian Billington, Martin Büchi, Melanie Caffrey, Ron Crisco, Lewis Cunningham, Dominic Delmolino, Sue Harper, Torben Holm, Connor McDonald, Arup Nanda, Stephan Petit, Michael Rosenblum, Robyn Sands, and Riyaj Shamsudeen
Apress®


Get up and running with PL/SQL programming —no experience necessary!
Oracle PL/SQL
FOR
DUMMIES
Author's Web site includes all code samples
A Reference for the Rest of Us!
FREE eTips at dummies.com
Dr. Paul Dorsey
Michael Rosenblum

Coming June:

PL/SQL Performance Tuning