



SAGE Computing Services

Customised Oracle Training Workshops and Consulting

Bulk Binds

Be A Bulk Binding Baron

Scott Wesley
Systems Consultant



Hasn't this been around forever?



Yep, you've heard of these before...

- Been around since 8i!
- Not used often enough
- Binding is the assignment of values to placeholders in SQL statements
- Bulk Binding is binding an array of values
 - in-bind : PL/SQL variable -> DB using INSERT/UPDATE
 - out-bind : DB -> PL/SQL variable using RETURNING
 - define : DB -> PL/SQL variable using SELECT/FETCH
- MULTISET

A quick word on *Collections*



Oracle Collections

- Collections are lists or sets of information, sometimes similar to 3GL arrays
- Invaluable – be comfortable with them
- Can be faster than SQL – cache frequent access in PGA
- Can consume lots of PGA memory
- Consider NOCOPY hint for formal parameter – but be very careful



Types of Collections

- Associative Arrays (aka index-by tables)
 - Only PL/SQL
 - Sparse
 - Can have negative indexes
 - Similar to hash table concept
 - Different datatypes for indexes
 - TYPE *type_name* IS TABLE OF *element_type* [NOT NULL]
INDEX BY [PLS_INTEGER | BINARY_INTEGER |
VARCHAR2(*size_limit*)];

```
TYPE emp_tab IS TABLE OF  
employees%ROWTYPE  
INDEX BY PLS_INTEGER;
```



Types of Collections

- Nested Tables
 - Can use in SQL
 - Unbounded
 - Good for set operations
 - TYPE *type_name* IS TABLE OF *element_type* [NOT NULL];

```
TYPE top_emp IS TABLE OF  
employees.employee_id%TYPE;
```

```
CREATE TYPE galaxy AS TABLE OF  
star_object;
```

Types of Collections

- VARRAY
 - Specify maximums
 - Dense
 - Can use in SQL
 - Most similar to 3GL array
 - TYPE *type_name* IS {VARRAY | VARYING ARRAY} (*size_limit*) OF *element_type* [NOT NULL];

```
TYPE Calendar IS VARRAY(366) OF DATE;
```

```
CREATE TYPE rainbow AS VARRAY(7) OF VARCHAR2(64);
```

```
CREATE TYPE solar_system AS VARRAY(8) OF  
planet_object;
```

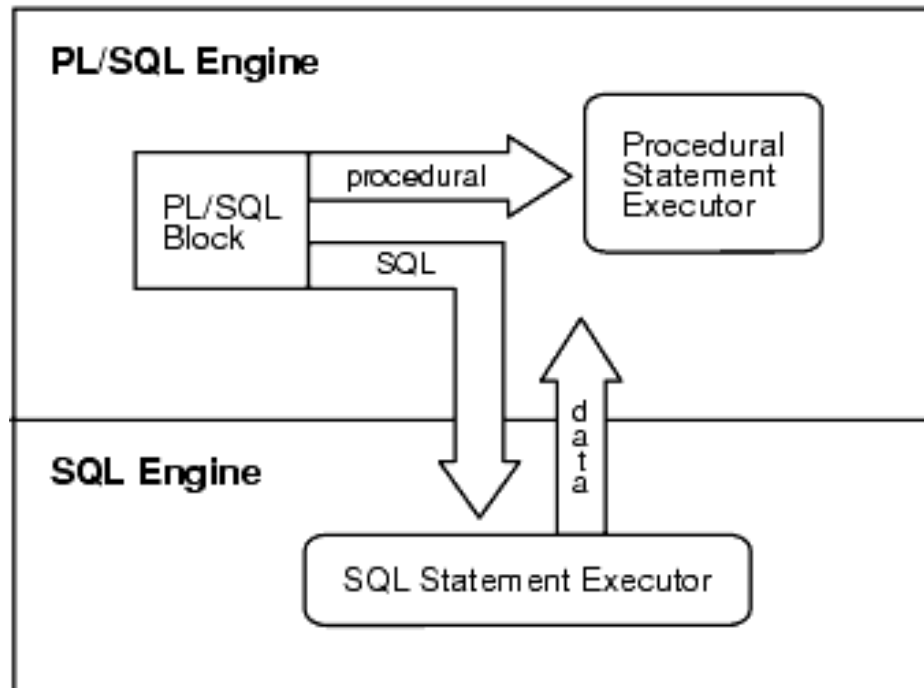


Back to the business of binds



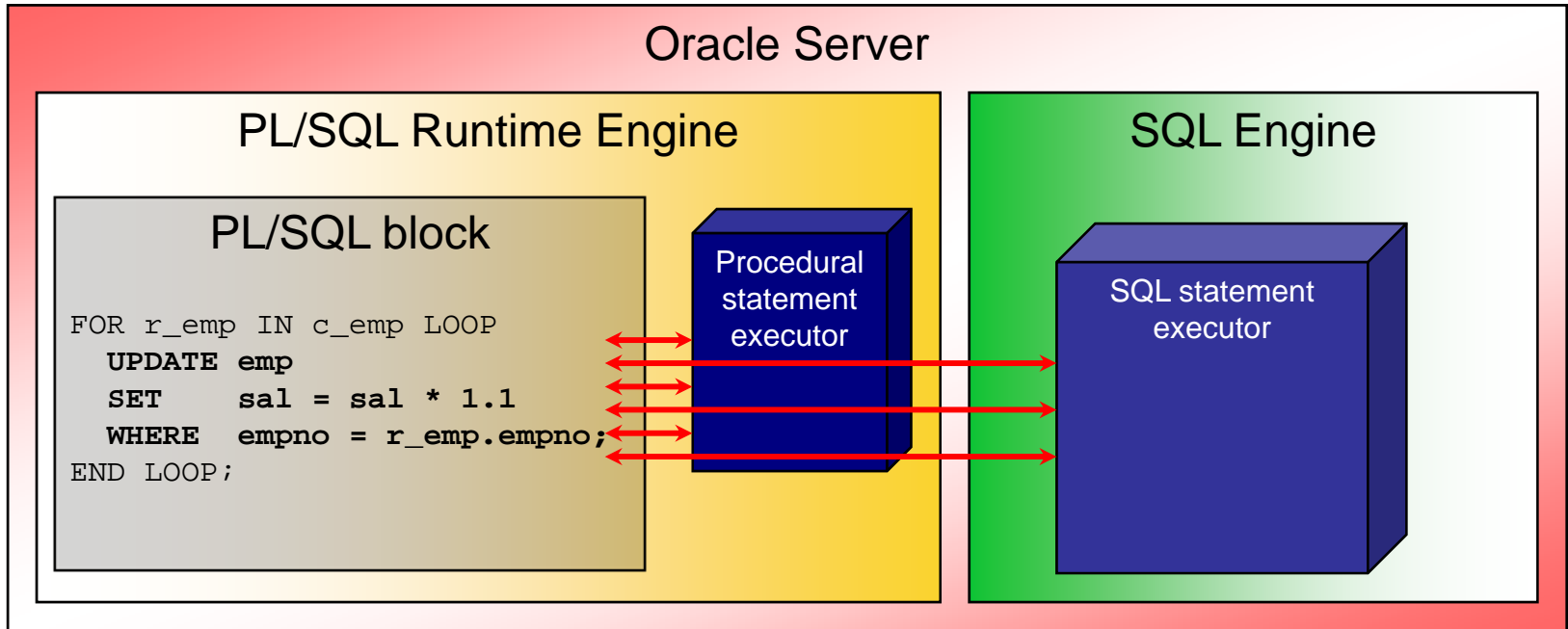
From the 8.1.7 documentation

- A little boring and makes you think...



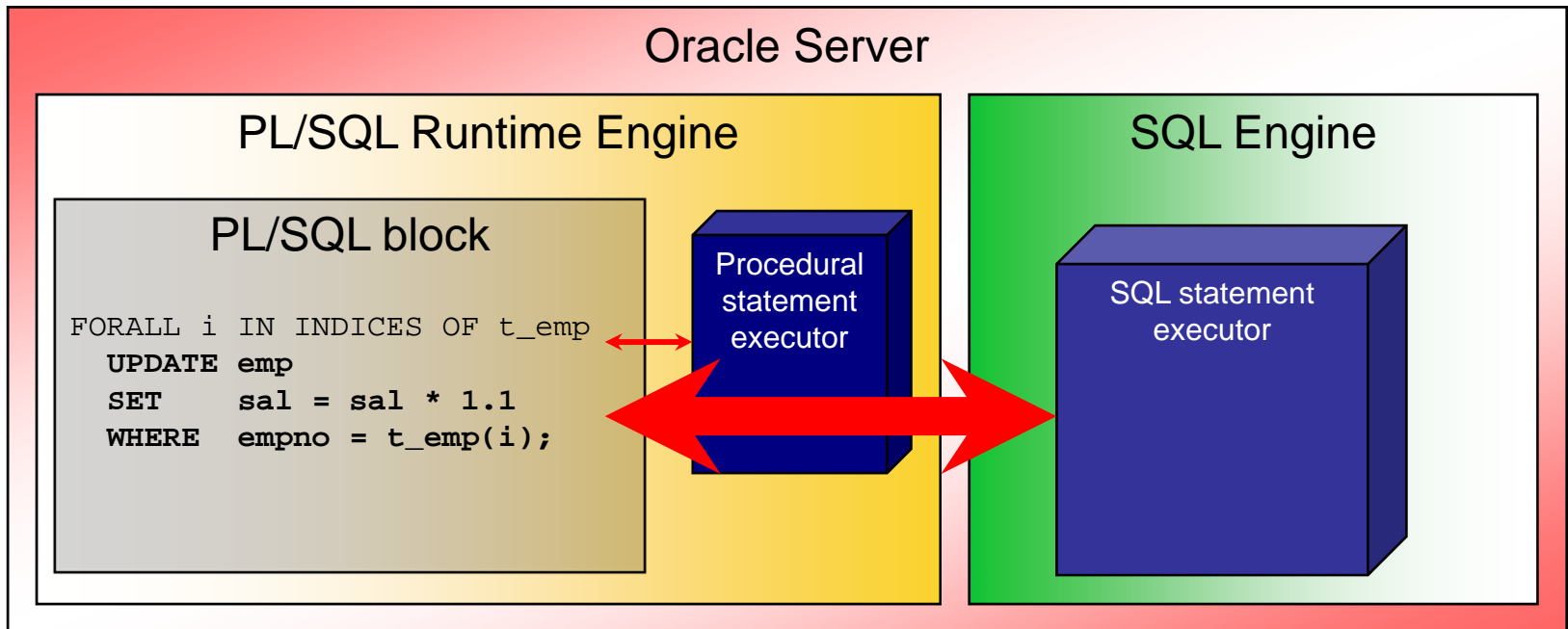
Conventional Bind

- Performance penalty for many context switches



Bulk Bind

- Much less overhead



When would you use it?



- No bulk bind!
- “If the DML statement affects four or more database rows, the use of bulk SQL can improve performance considerably” – straight from Oracle documentation
- Doesn't need to be massive set processing – need to consider memory issues if too large – `LIMIT`
- Reoccurring SQL statement in PL/SQL loop
 - To issue series of DML, replace `LOOP` with `FORALL`
 - Loop through result set and store values in memory with one operation with `BULK COLLECT`
 - Audit trigger

DML Bind Example

```
-- Slower method, running the UPDATE
-- statements within a regular loop
FOR i IN t_emp.FIRST..t_emp.LAST LOOP
    UPDATE emp_largish
    SET      sal = sal * 1.1
    WHERE    empno = t_emp(i);
END LOOP;
```

1.94 seconds

```
-- Efficient method, using a bulk bind on a VARRAY
FORALL i IN t_emp.FIRST..t_emp.LAST
    UPDATE emp_largish
    SET      sal = sal * 1.1
    WHERE    empno = t_emp(i);
```

1.40 seconds



Collect Example

```
-- Slower method, assigning each  
-- collection element within a loop.  
v_index := 0;  
FOR r_emp IN c_emp LOOP  
    v_index := v_index + 1;  
    t_emp(v_index).empno := r_emp.empno;  
    t_emp(v_index).ename := r_emp.ename;  
END LOOP;
```

0.19 seconds

```
-- Efficient method, using a bulk bind  
OPEN c_emp;  
FETCH c_emp BULK COLLECT INTO t_emp;  
CLOSE c_emp;
```

0.09 seconds



Example of Combination

```
-- Slower method, running update
-- statement within regular loop,
-- returning individual elements
FOR i IN t_emp.FIRST..t_emp.LAST LOOP
  UPDATE emp_largish
  SET      sal = sal * 1.1
  WHERE empno = t_emp(i)
  RETURNING sal BULK COLLECT
  INTO t_sal;
END LOOP;
```

0.75 seconds

0.29 seconds

When *wouldn't* you use it?



- Can be memory intensive, especially in PGA
- Do more work in SQL!
- If you only need to loop through it once, avoid memory overhead of storing a copy of the result set
- When you could just filter within query using `WHERE`, `EXISTS`, `INTERSECT` or `MINUS`
- When you haven't tested performance with & without!
- 10g silently does a form of bulk binds for you

I have a sparse collection



Uh oh...

```
DECLARE
  TYPE emp_tab IS TABLE OF emp.empno%TYPE INDEX BY PLS_INTEGER;
  t_emp emp_tab;
  CURSOR c_emp IS
    SELECT empno FROM emp;
BEGIN
  OPEN c_emp;
  FETCH c_emp BULK COLLECT INTO t_emp;
  CLOSE c_emp;

  -- ...do a bunch of processing, including
  -- something like:
  t_emp.DELETE(3);

  FORALL i IN t_emp.FIRST..t_emp.LAST
    UPDATE emp
      SET    sal = sal * 1.1
      WHERE  empno = t_emp(i);
END;
/
```

```
ORA-22160: element at index [3] does not exist
ORA-06512: at line 15
```

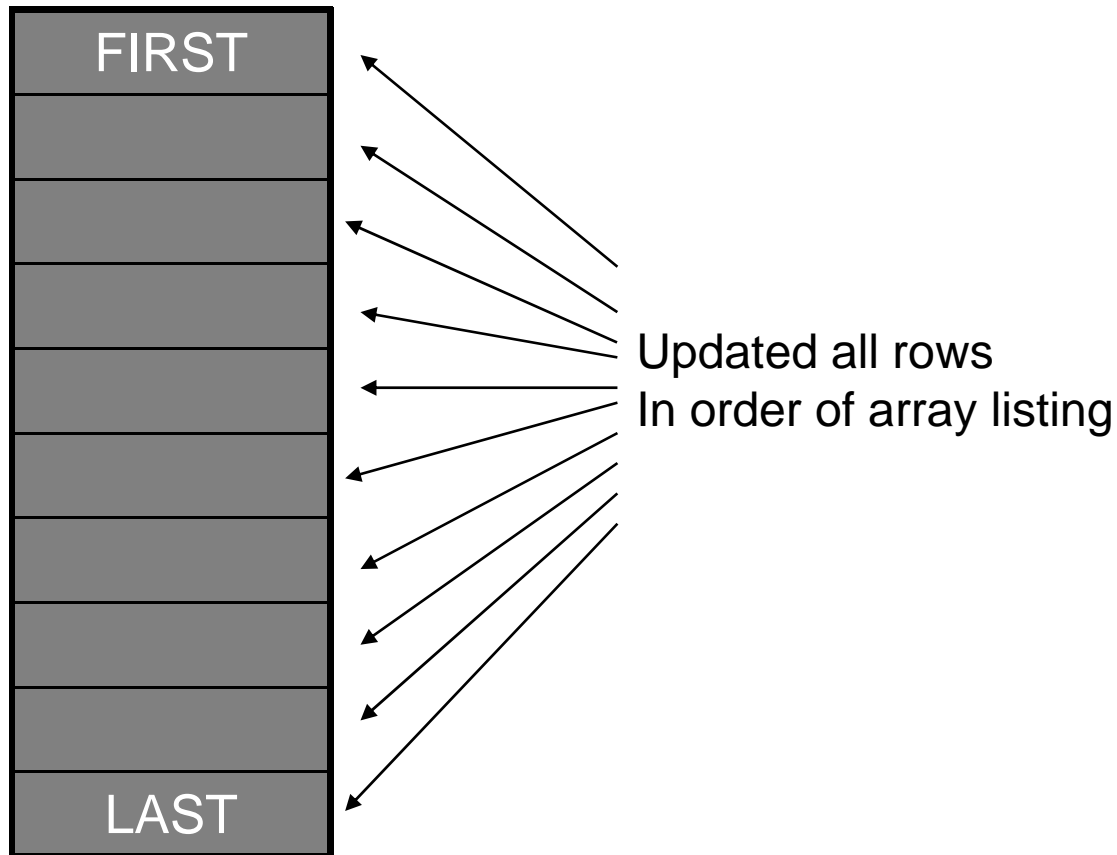
Processing Sparse Collections

- Introduced in 10g
- INDICES OF
 - allows iteration of all index values, not just upper and lower bound (`tab.FIRST..tab.LAST`)
- VALUES OF
 - Create PL/SQL table effectively indexing your collection
 - Process specific elements
 - Process in particular order
 - Process an element more than once



Regular

```
FORALL i IN t_emp.FIRST..t_emp.LAST  
  UPDATE emp  
  SET    sal = sal * 1.1  
  WHERE  empno = t_emp(i);
```



Regular

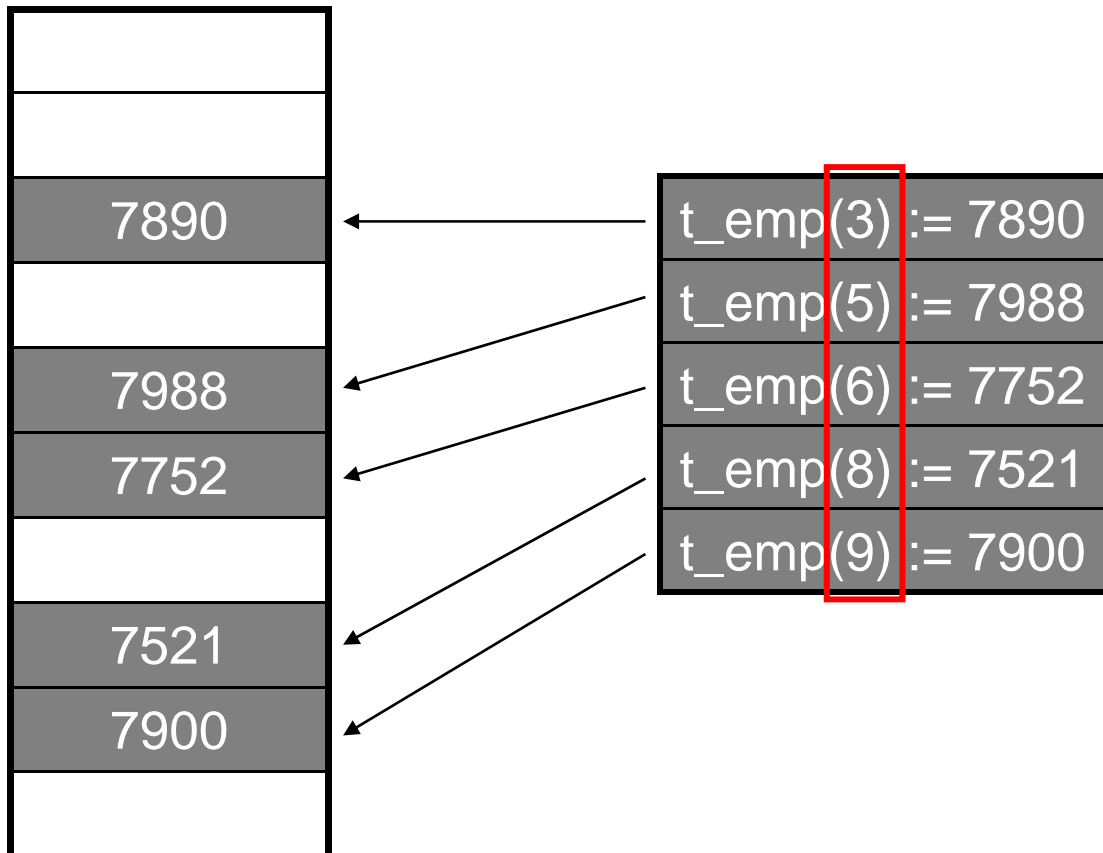
```
FORALL i IN t_emp.FIRST..t_emp.LAST  
  UPDATE emp  
  SET    sal = sal * 1.1  
  WHERE  empno = t_emp(i);
```



ORA-22160: element at index [3] does not exist

Premium

```
FORALL i IN INDICES OF t_emp  
  UPDATE emp  
  SET    comm = comm * 1.5  
  WHERE  empno = t_emp(i);
```



Updated rows within array
In order of array listing

High Octane!

DECLARE

TYPE emp_tab IS TABLE OF emp.empno%TYPE INDEX BY PLS_INTEGER;

TYPE numlist IS TABLE OF PLS_INTEGER INDEX BY BINARY_INTEGER;

t_emp emp_tab;

t_indexes numlist;

BEGIN

t_emp(3) := 7890;

t_emp(5) := 7988;

t_emp(6) := 7752;

t_emp(8) := 7521;

t_emp(9) := 7900;

t_indexes(-2) := 6;

t_indexes(0) := 3;

t_indexes(3) := 5;

t_indexes(10) := 6;

What if I have $9i$?



DECLARE

CREATE TABLE emp2 AS SELECT * FROM emp WHERE 1=0;

CREATE OR REPLACE TYPE emp_obj AS OBJECT

(empno NUMBER(4)
,ename VARCHAR2(10)
,job VARCHAR2(9)
,mgr NUMBER(4)
,hiredate DATE
,sal NUMBER(7,2)
,comm NUMBER(7,2)
,deptno NUMBER(2))

/

CREATE OR REPLACE TYPE t_emp_obj AS TABLE OF emp_obj

/

-- NO need for FORALL

INSERT INTO emp2

SELECT * FROM TABLE(CAST(t_emp AS t_emp_obj));

END;

/

Can I do this "bunch of processing" in bulk, outside the db?



```
DECLARE
  TYPE emp_tab IS TABLE OF emp.empno%TYPE INDEX BY PLS_INTEGER;
  t_emp emp_tab;
  CURSOR c_emp IS
    SELECT empno FROM emp;
BEGIN
  OPEN c_emp;
  FETCH c_emp BULK COLLECT INTO t_emp;
  CLOSE c_emp;
```

```
-- ... Or just those employees that exist in another collection
-- t_emp intersect with t_emp_leaders;
```

```
FORALL i IN t_emp.FIRST..t_emp.LAST
  UPDATE emp
  SET    sal = sal * 1.1
  WHERE  empno = t_emp(i);
END;
/
```

MULTISET Operations



Nested tables **ONLY**



DECLARE

```
TYPE colours IS TABLE OF VARCHAR2(64);  
comp1 colours;  
comp2 colours;  
result colours;
```

```
PROCEDURE show_table (p_table IN colours) IS  
BEGIN
```

```
    FOR i IN p_table.FIRST..p_table.LAST LOOP  
        DBMS_OUTPUT.PUT (p_table(i)||' ');  
    END LOOP;  
    DBMS_OUTPUT.NEW_LINE;
```

```
END;
```

BEGIN

```
comp1 := colours('Black','White','Red','Red');  
comp2 := colours('Black','Red','Yellow');
```

```
result := comp1 MULTISET UNION comp2;  
dbms_output.put_line ('multiset union is ');  
show_table(result);
```

```
END;
```

```
/
```

multiset union is

Black White Red Red Black Red Yellow



...

BEGIN

comp1 := colours('Black','White','Red','Red');

comp2 := colours('Black','Red','Yellow');

result := comp1 **MULTISET UNION DISTINCT** comp2;

dbms_output.put_line ('multiset union distinct is ');

show_table(result);

END;

/

multiset union distinct is

Black White Red Yellow

```
...  
BEGIN  
    comp1 := colours('Black','White','Red','Red');  
    comp2 := colours('Black','Red','Yellow');  
  
    result := comp1 MULTiset INTERSECT DISTINCT comp2;  
    dbms_output.put_line ('multiset intersect distinct is ');  
    show_table(result);  
  
END;  
/
```

```
multiset intersect distinct is  
Black Red
```

```
...  
BEGIN  
    comp1 := colours('Black','White','Red','Red');  
    comp2 := colours('Black','Red','Yellow');  
  
    result := comp1 MULTiset EXCEPT DISTINCT comp2;  
    dbms_output.put_line ('multiset except distinct is ');  
    show_table(result);  
  
END;  
/
```

```
multiset except distinct is  
White
```

```
...  
BEGIN  
    comp1 := colours('Black','White','Red','Red');  
  
    comp1 := comp1 MULTISET UNION DISTINCT comp1;  
    dbms_output.put_line ('self multiset intersect distinct is ');  
    show_table(comp1);  
  
END;  
/
```

```
self multiset union distinct is  
Black White Red
```



```
DECLARE
    TYPE colours IS TABLE OF VARCHAR2(64);
    t_colours colours := colours('Black','White','Red','Red');

    PROCEDURE show_table (p_table IN colours)
    ...
BEGIN
    DBMS_OUTPUT.PUT_LINE('Count:' || CARDINALITY(t_colours));
    DBMS_OUTPUT.PUT_LINE('Distinct:' || CARDINALITY(SET(t_colours)));
    IF t_colours IS NOT A SET THEN
        t_colours := SET(t_colours);
        show_table(t_colours);
    END IF;
END;
/
```

```
Count:4
Distinct:3
Black White Red
```



But wait, there's still more...



Equality

```
DECLARE
```

```
    TYPE colours IS TABLE OF VARCHAR2 (64);
```

```
    group1    colours := colours ('Black', 'White');
```

```
    group2    colours := colours ('White', 'Black');
```

```
    group3    colours := colours ('White', 'Red');
```

```
BEGIN
```



```
IF group1 = group2 THEN
```

```
    DBMS_OUTPUT.PUT_LINE ('Group 1 = Group 2');
```

```
ELSE
```

```
    DBMS_OUTPUT.PUT_LINE ('Group 1 != Group 2');
```

```
END IF;
```



```
IF group2 != group3 THEN
```

```
    DBMS_OUTPUT.PUT_LINE ('Group 2 != Group 3');
```

```
ELSE
```

```
    DBMS_OUTPUT.PUT_LINE ('Group 2 = Group 3');
```

```
END IF;
```

```
END;
```

```
/
```


Membership

```
DECLARE
  TYPE colours IS TABLE OF VARCHAR2(64);
  t_colours colours := colours('Black','White','Red','Red');
  v_colour VARCHAR2(64) := 'Black';
BEGIN
  IF v_colour MEMBER OF t_colours THEN
    DBMS_OUTPUT.PUT_LINE('Exists');
  ELSE
    DBMS_OUTPUT.PUT_LINE('Absent');
  END IF;
END;
/
```

Exists



Subset

```
DECLARE
    TYPE colours IS TABLE OF VARCHAR2(64);
    t_colours colours := colours('Black','White','Red','Yellow');
    t_colours2 colours := colours('Black','Red');
    t_colours3 colours := colours('Black','Blue');
BEGIN
    IF t_colours2 SUBMULTISET OF t_colours THEN
        DBMS_OUTPUT.PUT_LINE('2 Subset');
    ELSE
        DBMS_OUTPUT.PUT_LINE('2 Separate');
    END IF;

    IF t_colours3 SUBMULTISET OF t_colours THEN
        DBMS_OUTPUT.PUT_LINE('3 Subset');
    ELSE
        DBMS_OUTPUT.PUT_LINE('3 Separate');
    END IF;
END;
/
```

2 Subset

3 Separate



So what about these memory issues?



Boom!

```
DECLARE  
  TYPE t IS TABLE OF VARCHAR2(32) INDEX BY  
    PLS_INTEGER;  
  l_c t_c;  
BEGIN  
  SELECT LPAD  
    FROM  
  CONNECT  
END;
```

```
/  
decl  
*  
ERROR at line 1:  
ORA-04030: out of process memory trying to allocate  
16396 bytes (koh-kghu...,pl/s...,c2)  
ORA-06512: at line 5
```

Elapsed: 00:00:02.70

Limiting Data

- Prevent collections expanding with no limit

```
SELECT sal BULK COLLECT
INTO      t_emp
FROM      emp
WHERE ROWNUM <= 10;
```

```
WHERE ROWNUM BETWEEN 11 AND 20 -- Won't work
ORDER BY dbms_random.value; -- Can't use rownum
FROM emp SAMPLE(10) -- "Random"
```

```
-- Performance hit:
SELECT * FROM (
  SELECT ROW_NUMBER() OVER (ORDER BY empno)
         AS from_to_rank
... )
WHERE from_to_rank BETWEEN 1 AND 10
```

LIMIT Clause

- Process specified number of rows at a time

```
DECLARE
```

```
    v_rows    PLS_INTEGER := 10;
```

```
BEGIN
```

```
OPEN c_emp;
```

```
LOOP
```

```
    FETCH c_emp BULK COLLECT INTO t_emp LIMIT v_rows;
```

```
    EXIT WHEN t_emp.COUNT = 0;
```

```
    null; -- (Process information)
```

```
END LOOP;
```

```
CLOSE c_emp;
```



LIMIT Clause

- Process specified number of rows at a time

```
OPEN c_emp;
```

```
LOOP
```

```
    FETCH c_emp BULK COLLECT INTO t_emp LIMIT v_rows;
```

```
    EXIT WHEN t_emp.COUNT = 0; -- Here?
```

```
    null; -- (Process information)
```

```
    EXIT WHEN c_emp%NOTFOUND; -- or here?
```

```
END LOOP;
```

```
CLOSE c_emp;
```

```
EXCEPTION WHEN NO_DATA_FOUND
```

```
    -- Does this get raised?
```



`t_emp.COUNT = 0`

`** Count:10`

`CLARK`

`JAMES`

`MARTIN`

`MILLER`

`WARD`

`KING`

`ALLEN`

`ADAMS`

`SMITH`

`JONES`

`** Count:4`

`TURNER`

`BLAKE`

`SCOTT`

`FORD`

`** Count:0`

`LOOP`

`FETCH c_emp BULK COLLECT INTO t_emp LIMIT 10;`

`dbms_output.put_line('** Count:' || t_emp.COUNT);`

`EXIT WHEN t_emp.COUNT = 0;`

`dbms_output.put_line(t_emp(i).ename);`

`-- EXIT WHEN c_emp%NOTFOUND;`

`PL/SQL procedure successfully completed.`



WHEN c_emp%NOTFOUND

**** Count:10**

CLARK

JAMES

MARTIN

MILLER

WARD

KING

ALLEN

ADAMS

SMITH

JONES

**** Count:4**

TURNER

BLAKE

SCOTT

FORD

LOOP

```
FETCH c_emp BULK COLLECT INTO t_emp LIMIT 10;  
dbms_output.put_line('** Count:' || t_emp.COUNT);
```

```
-- EXIT WHEN t_emp.COUNT = 0;  
dbms_output.put_line(t_emp(i).ename);
```

```
EXIT WHEN c_emp%NOTFOUND;
```

PL/SQL procedure successfully completed.



Implicit Cursor != LIMIT

```
DECLARE
```

```
  TYPE emp_tab IS TABLE OF emp.empno%TYPE INDEX BY  
    PLS_INTEGER;
```

```
  t_emp emp_tab;
```

```
BEGIN
```

```
  SELECT empno
```

```
  BULK COLLECT INTO t_emp
```

```
  FROM emp
```

```
  LIMIT 10;
```

```
END;
```

```
/
```

```
  LIMIT 10
```

```
  *
```

```
ERROR at line 8:
```

```
ORA-06550: line 8, column 9:
```

```
PL/SQL: ORA-00933: SQL command not properly ended
```

```
ORA-06550: line 5, column 3:
```

```
PL/SQL: SQL Statement ignored
```

NO_DATA_FOUND

```
DECL DECLARE
V      TYPE emp_tab IS TABLE OF emp.sal%TYPE
BEG      INDEX BY PLS_INTEGER;
      t_emp emp_tab;
BEGIN
      SELECT sal BULK COLLECT
      INTO      t_emp
END      FROM      emp
/      WHERE 1=0;
DECL END;
*
/
ERR PL/SQL procedure successfully completed.
ORA-01403: no data found
ORA-06512: at line 4
```

For the bean counters...



Bulk Cursor Attributes

```
DECLARE
    TYPE dept_list IS TABLE OF NUMBER;
    t_dept dept_list := dept_list(10, 20, 30);
BEGIN
    FORALL i IN t_dept.FIRST..t_dept.LAST
        DELETE FROM emp WHERE deptno = t_dept(i);
    -- Each indice may update multiple rows
    dbms_output.put_line('Total rows affected: ' || SQL%ROWCOUNT);

    -- How many rows were affected by each delete statement?
    FOR j IN t_dept.FIRST..t_dept.LAST LOOP
        dbms_output.put_line('Dept ' || t_dept(j) ||
                               'rows:' || SQL%BULK_ROWCOUNT(j));
    END LOOP;
END;
/
```

Total rows affected: 14

Dept 10 rows:3

Dept 20 rows:5

Dept 30 rows:6

PL/SQL procedure successfully completed.



What if we have exceptions for a particular row?



```
CREATE TABLE emp2 AS SELECT * FROM emp;
```

```
DECLARE
```

```
BEGIN
```

```
FORALL i IN t_emp.FIRST..t_emp.LAST SAVE EXCEPTIONS
```

```
-- Will raise exception for sal of 1250 but not 1300
```

```
UPDATE emp2
```

```
SET      sal = sal/100
```

```
WHERE empno = t_emp(i);
```

```
dbms_output.put_line('Total rows affected: '  
|| SQL%ROWCOUNT);
```

```
EXCEPTION
```

```
WHEN e_dml_errors THEN -- Now we figure out what failed and why.
```

```
dbms_output.put_line('Total failed updates: ' || SQL%BULK_EXCEPTIONS.COUNT);
```

```
FOR j IN 1..SQL%BULK_EXCEPTIONS.COUNT LOOP
```

```
dbms_output.put_line('Error ' || j || ' for indice '  
|| SQL%BULK_EXCEPTIONS(j).ERROR_INDEX);
```

```
dbms_output.put_line('Error message is '  
|| SQLERRM(-SQL%BULK_EXCEPTIONS(j).ERROR_CODE));
```

```
END LOOP;
```

```
END;
```

```
Total failed updates: 2
```

```
Error 1 for indice 2
```

```
Error message is ORA-02290: check constraint (.) violated
```

```
Error 2 for indice 5
```

```
Error message is ORA-02290: check constraint (.) violated
```

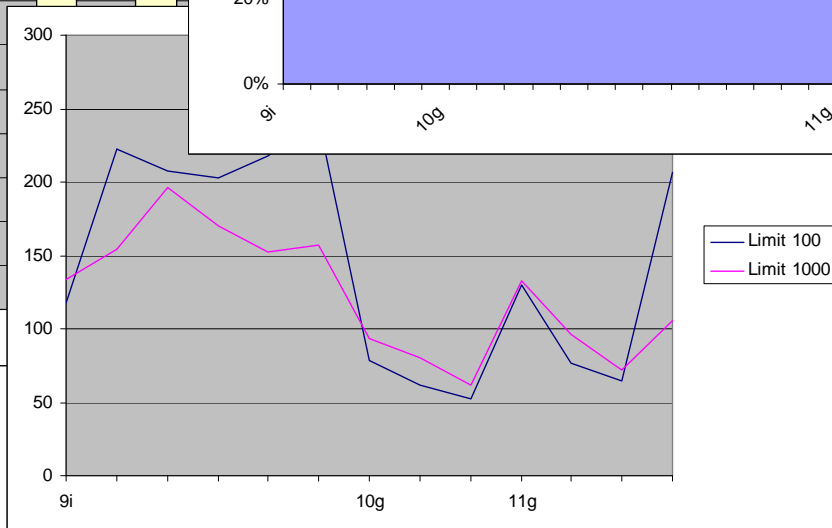
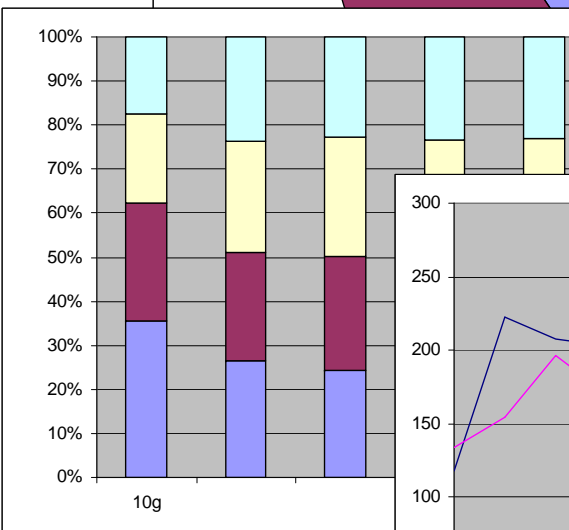
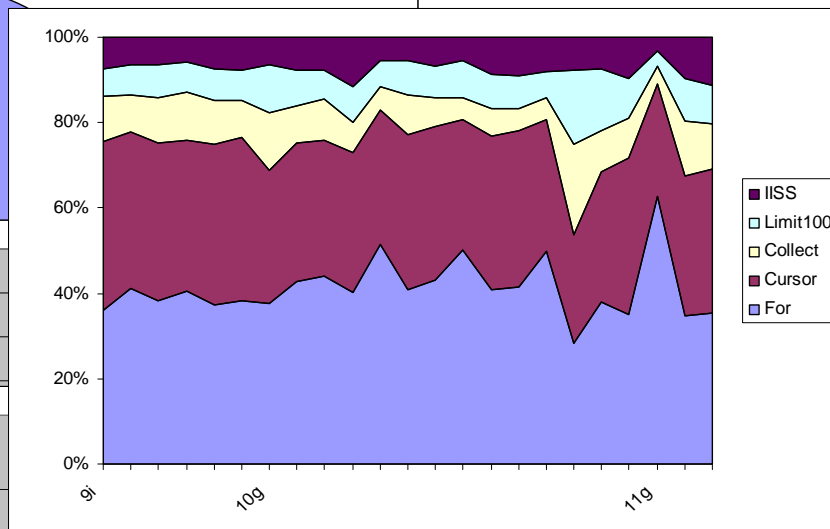
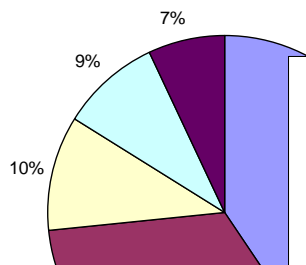
```
PL/SQL procedure successfully completed.
```

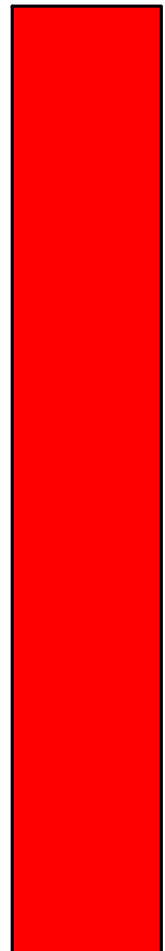
Bulk Exceptions

```
select empno, ename, sal from emp2 where  
       empno in (7934, 7782, 7839, 7788, 7900);
```

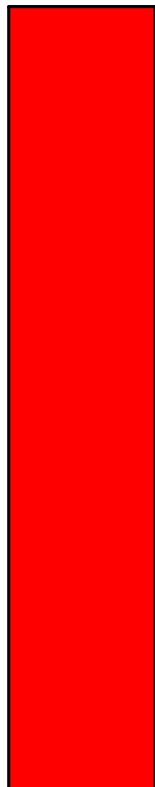
EMPNO	ENAME	SAL
7934	MILLER	13
7782	CLARK	2450
7839	KING	50
7788	SCOTT	30
7900	JAMES	950

5 rows selected.





FOR



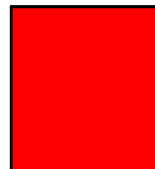
CURSOR



COLLECT

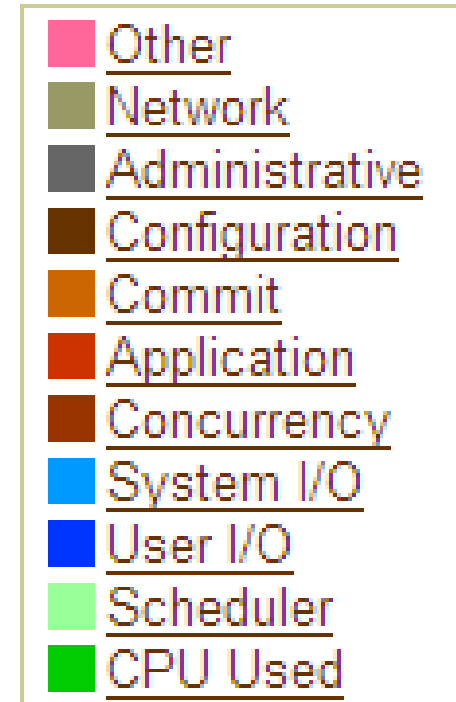
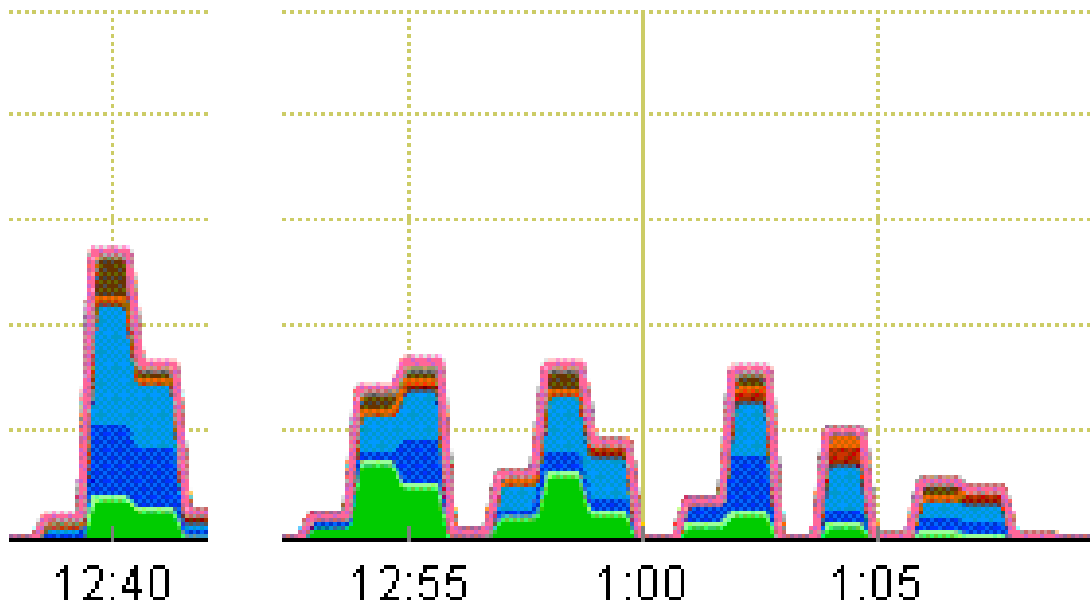


LIMIT100



IISS

Enterprise Manager



↑
Limit 100 vs 1000

↑
Separate runs of
For | Cursor | Collect | Limit | IISS

What's new?

- 8i
 - Bulk operations!
 - Collections indexed by BINARY_INTEGER only
- 9i
 - Bulk collect via dynamically opened ref cursor
- 9i Release 2
 - Bulk collect into collection of records
 - Index by VARCHAR2 & BINARY_INTEGER (or their subtypes)
- 10g
 - PL/SQL engine may decide to perform array fetching for you
 - INDICES OF/VALUES OF introduced
 - MULTISSET operations
- 11g
 - table(bulk_index).field is now supported at run-time
 - DBMS_SQL allows bulk binds using user-defined collection types

```
CREATE TABLE emp2 AS SELECT empno,ename,sal,   SYSTIMESTAMP ts
FROM emp WHERE 1=0;
```

```
DECLARE
```

```
    CURSOR sw IS SELECT empno, ename, sal FROM emp;
    TYPE t_sw IS TABLE OF sw%ROWTYPE;
    lt_sw t_sw;
```

```
BEGIN
```

```
    OPEN sw;
    FETCH sw BULK COLLECT INTO lt_sw;
    CLOSE sw;
    FORALL i IN lt_sw.FIRST..lt_sw.LAST
    INSERT INTO emp2
    VALUES (lt_sw(i).empno
            ,lt_sw(i).ename
            ,lt_sw(i).sal
            ,SYSTIMESTAMP);
```

```
END;
```

```
/
```

```
values (lt_sw(i).empno
      *
```

```
ERROR at line 11:
```

```
ORA-06550: line 11, column 9:
```

```
PLS-00436: implementation restriction: cannot reference fields of
      BULK In-BIND table of records
```

Some final words



**Read
appropriate
documentation**



READ THIS... TWICE!

- Tom Kyte
 - In search of the truth - Or Correlation is not Causation
 - http://asktom.oracle.com/pls/ask/download_file?p_file=3067171813508366601
 - *“If any paper you read tells you **what** but not **why**, regardless of its timestamp, I suggest you take its advice with a big grain of salt as it may be applicable **only to only specific cases and not applicable to you at all**. In fact, it could be something that was true in the past and not true anymore. **If they give you no way to find out, ignore it.**”*

Things change



It only takes one counter case to show something is not universally true



Rules of thumb without evidence, without ways to test them before implementing them, are dangerous, seriously dangerous

→ “Because a rule of thumb is a principle with broad application not intended to be strictly accurate or reliable for all situations”



SAGE Computing Services

**Customised Oracle Training Workshops and
Consulting**

HAVE YOU EVER THOUGHT YOU MIGHT LIKE A VULCAN MIND TRANSFER WITH THE SAGE EXPERTS?

The next best thing will be to attend Sage's course in advanced development techniques to be held early July.

Advanced Developers Workshop (3 days)

Presented by Penny Cookson and Chris Muir

- Advanced SQL techniques**
- Advanced PL/SQL techniques**
- Application performance tuning**
- The best of version 10g and a look at what's available in 11g.**

Contact Penny Cookson penny@sagecomputing.com.au to register.



SAGE Computing Services

Customised Oracle Training Workshops and Consulting

Questions and Answers?

Presentations are available from our website:

<http://www.sagecomputing.com.au>

enquiries@sagecomputing.com.au

scott.wesley@sagecomputing.com.au