|  |  |
| --- | --- |
| **General Information** | |
| Without the bulk bind, PL/SQLsends a SQL statement to the SQL engine for each record that is inserted, updated, or deleted leading to context switches that hurt performance. | |
| Syntax and Demo Directory | |  |  |  | | --- | --- | --- | | [BULK COLLECT](https://www.morganslibrary.org/reference/plsql/array_processing.html#apbc) | [FORALL INSERT](https://www.morganslibrary.org/reference/plsql/array_processing.html#apfa) | [PARTIAL COLLECTIONS](https://www.morganslibrary.org/reference/plsql/array_processing.html#appc) | | [Dynamic SQL Demos](https://www.morganslibrary.org/reference/plsql/array_processing.html#apds) | [FORALL MERGE](https://www.morganslibrary.org/reference/plsql/array_processing.html#apfm) | [SAVE EXCEPTIONS](https://www.morganslibrary.org/reference/plsql/array_processing.html#apeh) | | [Exception Handling](https://www.morganslibrary.org/reference/plsql/array_processing.html#apeh) | [FORALL UPDATE](https://www.morganslibrary.org/reference/plsql/array_processing.html#apfu) | [SPARSE COLLECTIONS](https://www.morganslibrary.org/reference/plsql/array_processing.html#apsc) | | [FETCH FIRST](https://www.morganslibrary.org/reference/plsql/array_processing.html#bcffc) | [IN INDICES OF](https://www.morganslibrary.org/reference/plsql/array_processing.html#apfa) | [Performance Demos](https://www.morganslibrary.org/reference/plsql/array_processing.html#appd) | | [FORALL DELETE](https://www.morganslibrary.org/reference/plsql/array_processing.html#apfd) | [LIMIT CLAUSE](https://www.morganslibrary.org/reference/plsql/array_processing.html#apbc) | [Things to Consider](https://www.morganslibrary.org/reference/plsql/array_processing.html#aptc) | |
|  | |
| **BULK COLLECT** | |
| BULK COLLECT Syntax and Demos | FETCH BULK COLLECT <cursor\_name> BULK COLLECT INTO <collection\_name> LIMIT <numeric\_expression>;  or  FETCH BULK COLLECT <cursor\_name> BULK COLLECT INTO <array\_name> LIMIT <numeric\_expression>; |
| set timing on  DECLARE  CURSOR a\_cur IS  SELECT program\_id  FROM airplanes; BEGIN   FOR cur\_rec IN a\_cur LOOP     NULL;   END LOOP; END; /  DECLARE  CURSOR a\_cur IS  SELECT program\_id  FROM airplanes;   TYPE myarray IS TABLE OF a\_cur%ROWTYPE;  cur\_array myarray; BEGIN   OPEN a\_cur;   LOOP     FETCH a\_cur BULK COLLECT INTO cur\_array LIMIT 100;     EXIT WHEN a\_cur%NOTFOUND;   END LOOP;   CLOSE a\_cur; END; /  DECLARE  CURSOR a\_cur IS  SELECT program\_id  FROM airplanes;   TYPE myarray IS TABLE OF a\_cur%ROWTYPE;  cur\_array myarray; BEGIN   OPEN a\_cur;   LOOP     FETCH a\_cur BULK COLLECT INTO cur\_array LIMIT 500;     EXIT WHEN a\_cur%NOTFOUND;   END LOOP;   CLOSE a\_cur; END; /  DECLARE  CURSOR a\_cur IS  SELECT program\_id  FROM airplanes;   TYPE myarray IS TABLE OF a\_cur%ROWTYPE;  cur\_array myarray; BEGIN   OPEN a\_cur;   LOOP     FETCH a\_cur BULK COLLECT INTO cur\_array LIMIT 1000;     EXIT WHEN a\_cur%NOTFOUND;   END LOOP;   CLOSE a\_cur; END; /  -- try with a LIMIT clause of 2500, 5000, and 10000. What do you see? |
| It used to be that you could not BULK COLLECT into an ARRAY OF RECORDS but that you could into a RECORD OF ARRAYS, as above.  This demo intentionally generated an error in versions prior to 11gR1. Familiarize yourself with the error and its message if in an earlier version. In 11gR1, and above, it is a powerful addition to the set of available tools. | CREATE OR REPLACE TYPE uw\_sel\_row AS OBJECT ( part\_num NUMBER, part\_name VARCHAR2(15)); /  CREATE OR REPLACE PROCEDURE fast\_way AUTHID CURRENT\_USER IS  TYPE uw\_sel\_tab IS TABLE OF uw\_sel\_row;  uw\_selection uw\_sel\_tab; BEGIN   SELECT uw\_sel\_row(part\_num, part\_name)   BULK COLLECT INTO uw\_selection   FROM parent;    FOR i IN 1..uw\_selection.COUNT LOOP     uw\_selection(i).part\_num := uw\_selection(i).part\_num \* 10;   END LOOP;    FORALL i IN 1..uw\_selection.COUNT   INSERT INTO child    VALUES   (uw\_selection(i).part\_num, uw\_selection(i).part\_name);   COMMIT; END fast\_way; / |
| Bulk Collect with DBMS\_SQL Data Types | CREATE TABLE t AS SELECT \* FROM all\_objects WHERE 1=0;  CREATE OR REPLACE PROCEDURE nrows\_at\_a\_time(p\_array\_size PLS\_INTEGER) AUTHID CURRENT\_USER IS  l\_owner          dbms\_sql.VARCHAR2\_table;  l\_object\_name    dbms\_sql.VARCHAR2\_table;  l\_subobject\_name dbms\_sql.VARCHAR2\_table;  l\_object\_id      dbms\_sql.NUMBER\_table;  l\_data\_object\_id dbms\_sql.NUMBER\_table;  l\_object\_type    dbms\_sql.VARCHAR2\_table;  l\_created        dbms\_sql.DATE\_table;  l\_last\_ddl\_time  dbms\_sql.DATE\_table;  l\_timestamp      dbms\_sql.VARCHAR2\_table;  l\_status         dbms\_sql.VARCHAR2\_table;  l\_temporary      dbms\_sql.VARCHAR2\_table;  l\_generated      dbms\_sql.VARCHAR2\_table;  l\_secondary      dbms\_sql.VARCHAR2\_table;   CURSOR c IS  SELECT \*  FROM all\_objects; BEGIN   OPEN c;   LOOP     FETCH c BULK COLLECT INTO     l\_owner, l\_object\_name, l\_subobject\_name, l\_object\_id,     l\_data\_object\_id, l\_object\_type, l\_created,      l\_last\_ddl\_time, l\_timestamp, l\_status, l\_temporary,      l\_generated, l\_secondary     LIMIT p\_array\_size;      FORALL i in 1 .. l\_owner.COUNT       INSERT INTO t       (owner, object\_name, subobject\_name, object\_id,        data\_object\_id, object\_type, created, last\_ddl\_time,        timestamp, status, temporary, generated, secondary)       VALUES       (l\_owner(i), l\_object\_name(i), l\_subobject\_name(i),        l\_object\_id(i), l\_data\_object\_id(i),         l\_object\_type(i), l\_created(i), l\_last\_ddl\_time(i),        l\_timestamp(i), l\_status(i), l\_temporary(i),        l\_generated(i), l\_secondary(i));     EXIT WHEN c%NOTFOUND;   END LOOP;   COMMIT;   CLOSE c; END nrows\_at\_a\_time; / |
| Fetch First Clause | Coming Soon In 12cR1 |
|  | |
| **FORALL** | |
| FORALL INSERT Syntax and Demos | FORALL <index\_name> IN <lower\_boundary> .. <upper\_boundary> <sql\_statement> SAVE EXCEPTIONS;  FORALL <index\_name> IN INDICES OF <collection> [BETWEEN <lower\_boundary> AND <upper\_boundary>] <sql\_statement> SAVE EXCEPTIONS;  FORALL <index\_name> IN INDICES OF <collection> VALUES OF <index\_collection> <sql\_statement> SAVE EXCEPTIONS; |
| CREATE TABLE servers2 AS SELECT \* FROM servers WHERE 1=2;  DECLARE  CURSOR s\_cur IS  SELECT \*  FROM servers;   TYPE fetch\_array IS TABLE OF s\_cur%ROWTYPE;  s\_array fetch\_array; BEGIN   OPEN s\_cur;   LOOP     FETCH s\_cur BULK COLLECT INTO s\_array LIMIT 1000;      FORALL i IN 1..s\_array.COUNT     INSERT INTO servers2 VALUES s\_array(i);      EXIT WHEN s\_cur%NOTFOUND;   END LOOP;   CLOSE s\_cur;   COMMIT; END; / |
| FORALL UPDATE: Basic | SELECT DISTINCT srvr\_id FROM servers2 ORDER BY 1;  DECLARE  TYPE myarray IS TABLE OF servers2.srvr\_id%TYPE  INDEX BY BINARY\_INTEGER;   d\_array myarray; BEGIN   d\_array(1) := 608;   d\_array(2) := 610;   d\_array(3) := 612;    FORALL i IN d\_array.FIRST .. d\_array.LAST   UPDATE servers2   SET srvr\_id = 0   WHERE srvr\_id = d\_array(i);    COMMIT; END; /  SELECT srvr\_id FROM servers2 WHERE srvr\_id = 0; |
| FORALL UPDATE: With SET ROW | CREATE TABLE servers3 AS SELECT \* FROM servers WHERE srvr\_id BETWEEN 501 AND 510;  SELECT \* FROM servers3;  DECLARE  CURSOR s\_cur IS  SELECT \*  FROM servers3;   TYPE s\_rec IS TABLE OF servers3%ROWTYPE  INDEX BY PLS\_INTEGER;  s\_array s\_rec;   -- uc = updated columns  TYPE uc\_cols\_rec IS RECORD(network\_id servers3.network\_id%TYPE);  TYPE uc\_array\_tt IS TABLE OF uc\_cols\_rec INDEX BY PLS\_INTEGER;  uc\_array uc\_array\_tt;   -- rid = row identifier  TYPE rid\_t IS TABLE OF servers3.srvr\_id%TYPE INDEX BY PLS\_INTEGER;  rid\_array rid\_t; BEGIN   OPEN s\_cur;   FETCH s\_cur BULK COLLECT INTO s\_array;   CLOSE s\_cur;    FOR i IN 1 .. s\_array.COUNT LOOP     -- load update key array     rid\_array(i) := s\_array(i).srvr\_id;     -- upate values     uc\_array(i).network\_id := s\_array(i).network\_id+1;   END LOOP;    FORALL i IN 1 .. s\_array.COUNT   UPDATE (SELECT network\_id FROM servers3 WHERE srvr\_id = rid\_array(i))   SET ROW = uc\_array(i);   COMMIT; END; /  SELECT \* FROM servers3; |
| FORALL DELETE | set serveroutput on  DECLARE  TYPE myarray IS TABLE OF servers2.srvr\_id%TYPE  INDEX BY BINARY\_INTEGER;   d\_array myarray; BEGIN   d\_array(1) := 614;   d\_array(2) := 615;   d\_array(3) := 616;    FORALL i IN d\_array.FIRST .. d\_array.LAST   DELETE servers2   WHERE srvr\_id = d\_array(i);    COMMIT;    FOR i IN d\_array.FIRST .. d\_array.LAST LOOP     dbms\_output.put\_line('Iteration #' || i || ' deleted ' ||     SQL%BULK\_ROWCOUNT(i) || ' rows.');   END LOOP; END; /  SELECT srvr\_id FROM servers2 WHERE srvr\_id IN (614, 615, 616); |
| FORALL MERGE | CREATE TABLE forall\_src ( rid NUMBER);  INSERT INTO forall\_src VALUES (1); COMMIT;  CREATE TABLE forall\_tgt ( rid NUMBER, ins VARCHAR2(1), upd VARCHAR2(1));  CREATE OR REPLACE PROCEDURE forall\_merge AUTHID CURRENT\_USER IS  TYPE ridVal IS TABLE OF forall\_tgt.rid%TYPE  INDEX BY BINARY\_INTEGER;  l\_data ridVal; BEGIN   SELECT rid BULK COLLECT INTO l\_data   FROM forall\_src;    FORALL i IN l\_data.FIRST .. l\_data.LAST   MERGE INTO forall\_tgt ft   USING (     SELECT rid     FROM forall\_src fs     WHERE fs.rid = l\_data(i)) al   ON (al.rid = ft.rid)   WHEN MATCHED THEN     UPDATE SET upd = 'U'   WHEN NOT MATCHED THEN     INSERT (rid, ins, upd)     VALUES (l\_data(i), 'I', NULL);    COMMIT; END forall\_merge; /  SELECT \* FROM forall\_src;  SELECT \* FROM forall\_tgt;  exec forall\_merge;  SELECT \* FROM forall\_tgt;  exec forall\_merge;  SELECT \* FROM forall\_tgt; |
|  | |
| **Partial Collections** | |
| Part of Collection Demo | CREATE TABLE test ( deptno  NUMBER(3,0), empname VARCHAR2(20));  INSERT INTO test VALUES (100, 'Morgan'); INSERT INTO test VALUES (200, 'Allen'); INSERT INTO test VALUES (101, 'Lofstrom'); INSERT INTO test VALUES (102, 'Havemeyer'); INSERT INTO test VALUES (202, 'Norgaard'); INSERT INTO test VALUES (201, 'Lewis'); INSERT INTO test VALUES (103, 'Scott'); INSERT INTO test VALUES (104, 'Foote'); INSERT INTO test VALUES (105, 'Townsend'); INSERT INTO test VALUES (106, 'Abedrabbo'); COMMIT;  SELECT \* FROM test;  CREATE OR REPLACE PROCEDURE collection\_part AUTHID CURRENT\_USER IS  TYPE NumList IS VARRAY(10) OF NUMBER;  depts NumList := NumList(100,200,101,102,202,201,103,104,105,106); BEGIN   FORALL j IN 4..7 -- use only part of varray   DELETE FROM test WHERE deptno = depts(j);   COMMIT; END collection\_part; /  SELECT \* FROM test; |
|  | |
| **Sparse Collections** | |
| A sparse collection is one from which elements have been deleted. | |
| Sparse Collection Demo using IN INDICES OF | ALTER TABLE child ADD CONSTRAINT uc\_child\_part\_num UNIQUE (part\_num) USING INDEX;  DECLARE  TYPE typ\_part\_name IS TABLE OF parent%ROWTYPE;  v\_part  typ\_part\_name; BEGIN   SELECT \*   BULK COLLECT INTO v\_part   FROM parent;    FOR rec IN 1 .. v\_part.LAST() LOOP     IF v\_part(rec).part\_name != 'Rectifier' THEN       v\_part.delete(rec);     END IF;   END LOOP;    FORALL i IN 1 .. v\_part.COUNT   INSERT INTO child   VALUES   v\_part(i);    COMMIT; END; /  DECLARE  TYPE typ\_part\_name IS TABLE OF parent%ROWTYPE;  v\_part  typ\_part\_name; BEGIN   SELECT \*   BULK COLLECT INTO v\_part   FROM parent;    FOR rec IN 1 .. v\_part.LAST LOOP     IF v\_part(rec).part\_name != 'Rectifier' THEN       v\_part.delete(rec);     END IF;   END LOOP;    FORALL idx IN INDICES OF v\_part   INSERT INTO child   VALUES   v\_part(idx);    COMMIT; END; /  SELECT COUNT(\*) FROM parent;  SELECT COUNT(\*) FROM child; |
| Using INDICES OF and VALUES OF with Non-Consecutive Index Values | CREATE TABLE valid\_orders ( cust\_name VARCHAR2(32), amount NUMBER(10,2));  CREATE TABLE big\_orders AS SELECT \* FROM valid\_orders WHERE 1 = 0;  CREATE TABLE rejected\_orders AS SELECT \* FROM valid\_orders WHERE 1 = 0;  DECLARE  -- collections to hold a set of customer names and amounts  SUBTYPE cust\_name IS valid\_orders.cust\_name%TYPE;  TYPE cust\_typ IS TABLe OF cust\_name;  cust\_tab cust\_typ;   SUBTYPE order\_amount IS valid\_orders.amount%TYPE;  TYPE amount\_typ IS TABLE OF NUMBER;  amount\_tab amount\_typ;   -- collections to point into the CUST\_TAB collection  TYPE index\_pointer\_t IS TABLE OF PLS\_INTEGER;   big\_order\_tab index\_pointer\_t := index\_pointer\_t();   rejected\_order\_tab index\_pointer\_t := index\_pointer\_t();  PROCEDURE setup\_data IS BEGIN   -- set up sample order data, with some invalid and 'big' orders   cust\_tab := cust\_typ('Company1', 'Company2', 'Company3',   'Company4', 'Company5');    amount\_tab := amount\_typ(5000.01, 0, 150.25, 4000.00, NULL); END setup\_data;  BEGIN   setup\_data;    dbms\_output.put\_line('--- Original order data ---');   FOR i IN 1..cust\_tab.LAST LOOP     dbms\_output.put\_line('Cust#' || i || ', '|| cust\_tab(i) ||     ': $'||amount\_tab(i));   END LOOP;    -- delete invalid orders (where amount is null or 0)   FOR i IN 1..cust\_tab.LAST LOOP     IF amount\_tab(i) is null or amount\_tab(i) = 0 THEN       cust\_tab.delete(i);       amount\_tab.delete(i);     END IF;   END LOOP;    dbms\_output.put\_line('---Data with deleted invalid orders---');    FOR i IN 1..cust\_tab.LAST LOOP     IF cust\_tab.EXISTS(i) THEN       dbms\_output.put\_line('Cust#' || i || ', ' || cust\_tab(i) ||       ': $'||amount\_tab(i));     END IF;   END LOOP;    -- since the subscripts of our collections are not consecutive,   -- we use use FORRALL...INDICES OF to iterate the subscripts   FORALL i IN INDICES OF cust\_tab   INSERT INTO valid\_orders   (cust\_name, amount)   VALUES   (cust\_tab(i), amount\_tab(i));    -- now process the order data differently extracting   --  2 subsets and storing each subset in a different table.   setup\_data; -- reinitialize the CUST\_TAB and AMOUNT\_TAB collections    FOR i IN cust\_tab.FIRST .. cust\_tab.LAST LOOP     IF amount\_tab(i) IS NULL OR amount\_tab(i) = 0 THEN       -- add a new element to the collection       rejected\_order\_tab.EXTEND;       -- record original collection subscript       rejected\_order\_tab(rejected\_order\_tab.LAST) := i;     END IF;      IF amount\_tab(i) > 2000 THEN       -- add a new element to the collection       big\_order\_tab.EXTEND;       -- record original collection subscript       big\_order\_tab(big\_order\_tab.LAST) := i;     END IF;   END LOOP;    -- run one DML statement on one subset of elements,   -- and another DML statement on a different subset.   FORALL i IN VALUES OF rejected\_order\_tab   INSERT INTO rejected\_orders VALUES (cust\_tab(i), amount\_tab(i));    FORALL i IN VALUES OF big\_order\_tab   INSERT INTO big\_orders VALUES (cust\_tab(i), amount\_tab(i));    COMMIT; END; /  -- verify that the correct order details were stored SELECT cust\_name "Customer", amount "Valid order amount" FROM valid\_orders;  SELECT cust\_name "Customer", amount "Big order amount" FROM big\_orders;  SELECT cust\_name "Customer", amount "Rejected order amount" FROM rejected\_orders; |
|  | |
| **Native Dynamic SQL** | |
| Native Dynamic SQL BULK COLLECT  Statement | DECLARE  sals dbms\_sql.number\_table; BEGIN   EXECUTE IMMEDIATE 'SELECT sal FROM emp'   BULK COLLECT INTO sals; END; / |
| Native Dynamic SQL FORALL Statement | CREATE TABLE tmp\_target AS SELECT rownum ID, table\_name, num\_rows FROM all\_tables WHERE rownum < 101;  DECLARE  TYPE NumList IS TABLE OF NUMBER;  rownos NumList;   TYPE NameList IS TABLE OF VARCHAR2(30);  tnames NameList; BEGIN   rownos := NumList(2,4,6,8,16);    FORALL i IN 1..5   EXECUTE IMMEDIATE 'UPDATE tmp\_target SET id = id \* 1.1   WHERE id = :1   RETURNING table\_name INTO :2'   USING rownos(i) RETURNING BULK COLLECT INTO tnames;    COMMIT;    FOR j IN 1..5 LOOP     dbms\_output.put\_line(tnames(j));   END LOOP; END; / |
| Combined BULK COLLECT and FORALL in a loop | -- while this demo does not require the limit clause it how to incorporate it into dynamic code  CREATE TABLE formulas ( formula\_no NUMBER(1), table\_name VARCHAR2(30), equation   CLOB);  CREATE TABLE calc\_results ( formula\_no NUMBER, result\_val NUMBER);  INSERT INTO formulas VALUES (1, 'SERVERS', 'AVG(latitude)'); INSERT INTO formulas VALUES (2, 'SERVERS', 'AVG(longitude)'); INSERT INTO formulas VALUES (3, 'AIRPLANES', 'COUNT(DISTINCT customer\_id)'); COMMIT;  col equation format a50 SELECT \* FROM formulas;  DECLARE  TYPE form\_t IS TABLE OF formulas%ROWTYPE;  form\_a form\_t;   TYPE rslt\_t IS TABLE OF calc\_results%ROWTYPE;  rslt\_a rslt\_t;   cSQL\_Cur   INTEGER;  cur\_var    SYS\_REFCURSOR; BEGIN   FOR tname\_rec IN (SELECT DISTINCT table\_name FROM formulas) LOOP -- **1**     BEGIN       SELECT formula\_no, table\_name, equation       BULK COLLECT INTO form\_a       FROM formulas       WHERE table\_name = tname\_rec.table\_name;     END;      DECLARE      vSQL\_Stmt   CLOB;      cRetVal     INTEGER;     BEGIN       vSQL\_Stmt := 'SELECT ';       FOR j IN 1 .. form\_a.COUNT LOOP    -- **2**: build the statement         vSQL\_Stmt := vSQL\_Stmt || form\_a(j).equation || ',';       END LOOP;                          -- **2**: build the statement       vSQL\_Stmt := TRIM(TRAILING ',' FROM vSQL\_Stmt);       vSQL\_Stmt := vSQL\_Stmt || ' FROM ' || tname\_rec.table\_name;        cSQL\_Cur := dbms\_sql.open\_cursor;       dbms\_sql.parse(cSQL\_Cur, vSQL\_Stmt, dbms\_sql.NATIVE);       cRetVal := dbms\_sql.execute(cSQL\_Cur);       cur\_var := dbms\_sql.to\_refcursor(cSQL\_Cur);     END;     DECLARE      results    dbms\_sql.number\_table;      cBatchSize CONSTANT POSITIVEN := 1000;     BEGIN       LOOP         FETCH cur\_var BULK COLLECT INTO results LIMIT cBatchsize;         FOR j in 1 .. results.COUNT LOOP           dbms\_output.put\_line(results(j));         END LOOP;         EXIT WHEN results.COUNT < cBatchSize;      END LOOP;     END;   END LOOP; -- **1**: outer loop retrieving distinct table names   CLOSE cur\_var; END; / |
|  | |
| **Exception Handling** | |
| Bulk Collection Exception Handling | CREATE TABLE tmp\_target AS SELECT table\_name, num\_rows FROM all\_tables WHERE 1=2;  ALTER TABLE tmp\_target ADD CONSTRAINT cc\_num\_rows CHECK (num\_rows > 0);   |  | | --- | | CREATE OR REPLACE PROCEDURE forall\_errors AUTHID CURRENT\_USER IS  TYPE myarray IS TABLE OF tmp\_target%ROWTYPE; l\_data myarray;  CURSOR c IS SELECT table\_name, num\_rows FROM all\_tables;  errors PLS\_INTEGER;  array\_dml EXCEPTIONS; PRAGMA EXCEPTION\_INIT(dml\_errors, -24381);   BEGIN   OPEN c;   LOOP     FETCH c BULK COLLECT INTO l\_data LIMIT 100;      -- SAVE EXCEPTIONS means don't stop if some DELETES fail     FORALL i IN 1..l\_data.COUNT SAVE EXCEPTIONS     INSERT INTO tmp\_target VALUES l\_data(i);      -- if any errors occurred during the FORALL SAVE EXCEPTIONS,     -- a single exception is raised when the statement completes.     EXIT WHEN c%NOTFOUND;   END LOOP;   COMMIT; EXCEPTION   WHEN dml\_errors THEN     errors := SQL%BULK\_EXCEPTIONS.COUNT;     dbms\_output.put\_line('Number of DELETE statements that     failed: ' || errors);      FOR i IN 1 .. errors LOOP       dbms\_output.put\_line('Error #' || i || ' at '|| 'iteration       #' || SQL%BULK\_EXCEPTIONS(i).ERROR\_INDEX);       dbms\_output.put\_line('Error message is ' ||       SQLERRM(-SQL%BULK\_EXCEPTIONS(i).ERROR\_CODE));     END LOOP;   WHEN OTHERS THEN     RAISE; END forall\_errors; / |   SQL> exec forall\_errors;  SQL>  SELECT \* FROM tmp\_target; |
| Exception Handling Demo | CREATE OR REPLACE PROCEDURE array\_exceptions AUTHID CURRENT\_USER IS  -- cursor for processing load\_errors  CURSOR le\_cur IS  SELECT \*  FROM load\_errors  FOR UPDATE;   TYPE myarray IS TABLE OF test%ROWTYPE;  l\_data myarray;   CURSOR c IS  SELECT sub\_date, cust\_account\_id, carrier\_id, ticket\_id, upd\_date  FROM stage  FOR UPDATE SKIP LOCKED;   errors PLS\_INTEGER;   cai   test.cust\_account\_id%TYPE;  cid   test.carrier\_id%TYPE;  ecode NUMBER;  iud   stage.upd\_date%TYPE;  sd    test.sub\_date%TYPE;  tid   test.ticket\_id%TYPE;  upd   test.upd\_date%TYPE; BEGIN   OPEN c;   LOOP     FETCH c BULK COLLECT INTO l\_data LIMIT 50000;      FORALL i IN 1..l\_data.COUNT SAVE EXCEPTIONS     INSERT INTO test VALUES l\_data(i);      EXIT WHEN c%NOTFOUND;   END LOOP;   COMMIT; -- exit here when no exceptions are raised EXCEPTION   WHEN OTHERS THEN     -- get the number of errors in the exception array     errors := SQL%BULK\_EXCEPTIONS.COUNT;      -- insert all exceptions into the load\_errors table     FOR j IN 1 .. errors LOOP       ecode := SQL%BULK\_EXCEPTIONS(j).ERROR\_CODE;       sd := TRUNC(l\_data(SQL%BULK\_EXCEPTIONS(j).ERROR\_INDEX).sub\_date);       cai := l\_data(SQL%BULK\_EXCEPTIONS(j).ERROR\_INDEX).cust\_account\_id;       cid := l\_data(SQL%BULK\_EXCEPTIONS(j).ERROR\_INDEX).carrier\_id;       tid := l\_data(SQL%BULK\_EXCEPTIONS(j).ERROR\_INDEX).ticket\_id;        INSERT INTO load\_errors       (error\_code, sub\_date, cust\_account\_id, carrier\_id, ticket\_id)       VALUES       (ecode, sd, cai, cid, tid);     END LOOP;      -- for each record in load\_errors process those that can     -- be handled and delete them after successful handling     FOR le\_rec IN le\_cur LOOP       IF le\_rec.error\_code = 1 THEN         SELECT upd\_date         INTO iud         FROM test         WHERE cust\_account\_id = le\_rec.cust\_account\_id         AND carrier\_id = le\_rec.carrier\_id         AND ticket\_id = le\_rec.ticket\_id;          IF iud IS NULL THEN           RAISE;         ELSIF iud < le\_rec.upd\_date THEN           UPDATE test           SET upd\_date = le\_rec.upd\_date           WHERE sub\_date = le\_rec.sub\_date           AND cust\_account\_id = le\_rec.cust\_account\_id           AND carrier\_id = le\_rec.carrier\_id           AND ticket\_id = le\_rec.ticket\_id;         ELSE           RAISE;         END IF;       END IF;     END LOOP;     COMMIT; -- exits here when any existing found END array\_exceptions; / |
|  | |
| **Performance Demos** | |
| Performance Comparison | CREATE TABLE t1 (pnum INTEGER, pname VARCHAR2(15)); CREATE TABLE t2 AS SELECT \* FROM t1;  CREATE OR REPLACE PROCEDURE perf\_compare(iterations PLS\_INTEGER) AUTHID CURRENT\_USER IS  TYPE NumTab IS TABLE OF t1.pnum%TYPE INDEX BY PLS\_INTEGER;  TYPE NameTab IS TABLE OF t1.pname%TYPE INDEX BY PLS\_INTEGER;  pnums  NumTab;  pnames NameTab;   a INTEGER;   b INTEGER;   c INTEGER;  BEGIN   FOR j IN 1..iterations LOOP -- load index-by tables     pnums(j) := j;     pnames(j) := 'Part No. ' || TO\_CHAR(j);   END LOOP;    a := dbms\_utility.get\_time;    FOR i IN 1..iterations LOOP -- use FOR loop     INSERT INTO t1 VALUES (pnums(i), pnames(i));   END LOOP;    b := dbms\_utility.get\_time;    FORALL i IN 1 .. iterations -- use FORALL statement   INSERT INTO t2 VALUES (pnums(i), pnames(i));    c := dbms\_utility.get\_time;    dbms\_output.put\_line('Execution Time (secs)');   dbms\_output.put\_line('---------------------');   dbms\_output.put\_line('FOR loop: ' || TO\_CHAR((b - a)/100));   dbms\_output.put\_line('FORALL: ' || TO\_CHAR((c - b)/100));   COMMIT; END perf\_compare; /  set serveroutput on  exec perf\_compare(500); exec perf\_compare(5000); exec perf\_compare(50000); |
| Bulk Collection Demo Table | CREATE TABLE parent ( part\_num  NUMBER, part\_name VARCHAR2(15));  CREATE TABLE child AS SELECT \* FROM parent; |
| Create And Load Demo Data | DECLARE  k parent.part\_name%TYPE := 'Transducer'; BEGIN   FOR i IN 1 .. 200000   LOOP     SELECT DECODE(k, 'Transducer', 'Rectifier',     'Rectifier', 'Capacitor',     'Capacitor', 'Knob',     'Knob', 'Chassis',     'Chassis', 'Transducer')     INTO k     FROM dual;      INSERT INTO parent VALUES (i, k);   END LOOP;   COMMIT; END; /  SELECT COUNT(\*) FROM parent; SELECT COUNT(\*) FROM child; |
| Slow Way | CREATE OR REPLACE PROCEDURE slow\_way AUTHID CURRENT\_USER IS BEGIN   FOR r IN (SELECT \* FROM parent) LOOP     -- modify record values     r.part\_num := r.part\_num \* 10;     -- store results     INSERT INTO child     VALUES     (r.part\_num, r.part\_name);   END LOOP;   COMMIT; END slow\_way; /  set timing on  exec slow\_way -- 07.71 |
| Fast Way 1  Fetch into user defined array | CREATE OR REPLACE PROCEDURE fast\_way AUTHID CURRENT\_USER IS  TYPE myarray IS TABLE OF parent%ROWTYPE;  l\_data myarray;   CURSOR r IS  SELECT part\_num, part\_name  FROM parent;   BatchSize CONSTANT POSITIVE := 1000; BEGIN   OPEN r;   LOOP     FETCH r BULK COLLECT INTO l\_data LIMIT BatchSize;      FOR j IN 1 .. l\_data.COUNT LOOP       l\_data(j).part\_num := l\_data(j).part\_num \* 10;     END LOOP;      FORALL i IN 1..l\_data.COUNT     INSERT INTO child VALUES l\_data(i);      EXIT WHEN l\_data.COUNT < BatchSize;   END LOOP;   COMMIT;   CLOSE r; END fast\_way; /  set timing on  exec fast\_way -- 00.50  set timing off  SELECT 7.71/0.50 FROM dual; |
| Fast Way 2  Fetch into user defined PL/SQL table | CREATE OR REPLACE PROCEDURE fast\_way AUTHID CURRENT\_USER IS  TYPE PartNum IS TABLE OF parent.part\_num%TYPE  INDEX BY BINARY\_INTEGER;   pnum\_t PartNum;   TYPE PartName IS TABLE OF parent.part\_name%TYPE  INDEX BY BINARY\_INTEGER;   pnam\_t PartName; BEGIN   SELECT part\_num, part\_name   BULK COLLECT INTO pnum\_t, pnam\_t   FROM parent;    FOR i IN pnum\_t.FIRST .. pnum\_t.LAST LOOP     pnum\_t(i) := pnum\_t(i) \* 10;   END LOOP;    FORALL i IN pnum\_t.FIRST .. pnum\_t.LAST   INSERT INTO child   (part\_num, part\_name)   VALUES   (pnum\_t(i), pnam\_t(i));   COMMIT; END fast\_way; /  set timing on  exec fast\_way -- 0.62 |
| Fast Way 3  Fetch into DBMS\_SQL defined array | CREATE OR REPLACE PROCEDURE fast\_way AUTHID CURRENT\_USER IS  TYPE parent\_rec IS RECORD (  part\_num   dbms\_sql.number\_table,  part\_name  dbms\_sql.varchar2\_table);   p\_rec parent\_rec;   CURSOR c IS  SELECT part\_num, part\_name FROM parent;   l\_done BOOLEAN; BEGIN   OPEN c;   LOOP     FETCH c BULK COLLECT INTO p\_rec.part\_num, p\_rec.part\_name     LIMIT 500;     l\_done := c%NOTFOUND;      FOR i IN 1 .. p\_rec.part\_num.COUNT LOOP       p\_rec.part\_num(i) := p\_rec.part\_num(i) \* 10;     END LOOP;      FORALL i IN 1 .. p\_rec.part\_num.COUNT     INSERT INTO child     (part\_num, part\_name)     VALUES     (p\_rec.part\_num(i), p\_rec.part\_name(i));      EXIT WHEN (l\_done);   END LOOP;   COMMIT;   CLOSE c; END fast\_way; /  set timing on  exec fast\_way -- 0.51 |
| Fast Way 4  Effect of triggers on performance of cursor loops vs. array processing | TRUNCATE TABLE child;  set timing on  exec slow\_way; exec fast\_way;  set timing off  TRUNCATE TABLE child;  CREATE OR REPLACE TRIGGER bi\_child BEFORE INSERT ON child FOR EACH ROW BEGIN   NULL; END bi\_child; /  set timing on  exec slow\_way; -- elapsed: 00:05:54.36  exec fast\_way; -- elapsed: 00:00:01.96 |
| Fast Way 5  Insert into multiple tables | TRUNCATE TABLE child;  RENAME child TO child1;  CREATE TABLE child2 AS SELECT \* FROM child1;   |  | | --- | | CREATE OR REPLACE PROCEDURE fast\_way AUTHID CURRENT\_USER IS  TYPE myarray IS TABLE OF parent%ROWTYPE;  l\_data myarray;   CURSOR r IS  SELECT part\_num, part\_name FROM parent; BEGIN   OPEN r;   LOOP     FETCH r BULK COLLECT INTO l\_data LIMIT 1000;      FOR j IN 1 .. l\_data.COUNT LOOP       l\_data(j).part\_num := l\_data(j).part\_num \* 10;     END LOOP;      FORALL i IN 1..l\_data.COUNT     INSERT INTO child1 VALUES l\_data(i);      FORALL i IN 1..l\_data.COUNT     INSERT INTO child2 VALUES l\_data(i);      EXIT WHEN r%NOTFOUND;   END LOOP;   COMMIT;   CLOSE r; END fast\_way; / |   set timing on  exec fast\_way |
| Old Way vs Fast Way with Exception Handling | CREATE TABLE tsource AS SELECT \* FROM dba\_tables;  INSERT INTO tsource SELECT \* FROM tsource WHERE table\_name = 'DUAL';  SELECT COUNT(\*) FROM tsource;  SELECT table\_name, COUNT(\*) FROM tsource GROUP BY table\_name HAVING COUNT(\*) > 1;  CREATE TABLE ttarget AS SELECT \* FROM tsource WHERE 1=2;  ALTER TABLE ttarget ADD CONSTRAINT pk\_ttarget PRIMARY KEY (table\_name);  CREATE OR REPLACE PROCEDURE old\_way AUTHID DEFINER IS BEGIN   FOR orec IN (SELECT \* FROM tsource) LOOP     INSERT INTO ttarget VALUES orec;   END LOOP;   COMMIT; END old\_way; /  CREATE OR REPLACE PROCEDURE old\_way AUTHID DEFINER IS BEGIN   FOR orec IN (SELECT \* FROM tsource) LOOP     BEGIN       INSERT INTO ttarget VALUES orec;     EXCEPTION       WHEN OTHERS THEN         NULL;     END;   END LOOP;   COMMIT; END old\_way; /  exec old\_way; exec old\_way; exec old\_way; exec old\_way;  TRUNCATE TABLE ttarget;  CREATE OR REPLACE PROCEDURE new\_way AUTHID DEFINER IS  TYPE myarray IS TABLE OF tsource%ROWTYPE;  l\_data myarray;   CURSOR ocur IS  SELECT \*  FROM tsource; BEGIN   OPEN ocur;   FETCH ocur BULK COLLECT INTO l\_data LIMIT 1000;   FORALL i IN 1..l\_data.COUNT SAVE EXCEPTIONS   INSERT INTO ttarget VALUES l\_data(i);   COMMIT;   CLOSE ocur; EXCEPTION   WHEN OTHERS THEN     NULL; END new\_way; /  exec new\_way; exec new\_way; exec new\_way; exec new\_way; |
|  | |
| **Things To Consider** | |
| This code demonstrates a technique that is syntactically valid. But, as you will see in the second example, removes all of the benefits of using FORALL. | CREATE TABLE airplanes2 AS SELECT \* FROM airplanes WHERE 1=2;  DECLARE  TYPE myarray IS TABLE OF airplanes2%ROWTYPE;  l\_data myarray;   CURSOR r IS  SELECT \*  FROM airplanes;   BatchSize CONSTANT POSITIVE := 1000; BEGIN   OPEN r;   LOOP     FETCH r BULK COLLECT INTO l\_data LIMIT BatchSize;      FORALL i IN 1..l\_data.COUNT     INSERT INTO airplanes2 VALUES l\_data(i);      EXIT WHEN l\_data.COUNT < BatchSize;   END LOOP;   COMMIT;   CLOSE r; END fast\_way; /   DECLARE  TYPE myarray IS TABLE OF airplanes2%ROWTYPE;  l\_data myarray;   CURSOR r IS  SELECT \*  FROM airplanes;   BatchSize CONSTANT POSITIVE := 1000; BEGIN   OPEN r;   LOOP     FETCH r BULK COLLECT INTO l\_data LIMIT BatchSize;      FORALL i IN 1..l\_data.COUNT     INSERT INTO airplanes2     (customer\_id, line\_number)     SELECT l\_data(i).customer\_id, l\_data(i).line\_number     FROM dual;      EXIT WHEN l\_data.COUNT < BatchSize;   END LOOP;   COMMIT;   CLOSE r; END fast\_way; / |
| One way to aggregate raw data. In this case summing the values of line\_number | CREATE TABLE airplanes2 AS SELECT \* FROM airplanes;  INSERT INTO airplanes2 SELECT \* FROM airplanes2;  INSERT INTO airplanes2 SELECT \* FROM airplanes2;  set serveroutput on  DECLARE  TYPE myarray IS TABLE OF ap%ROWTYPE;  l\_data myarray;  a\_data myarray := myarray();   CURSOR r IS  SELECT \*  FROM ap;   BatchSize CONSTANT POSITIVE := 100; BEGIN   a\_data.extend(1);   a\_data(1).line\_number := 0;   OPEN r;   LOOP     FETCH r BULK COLLECT INTO l\_data LIMIT BatchSize;      FOR j IN 1 .. l\_data.COUNT LOOP       a\_data(1).line\_number := a\_data(1).line\_number + l\_data(j).line\_number;     END LOOP;     EXIT WHEN l\_data.COUNT < BatchSize;   END LOOP;   dbms\_output.put\_line(a\_data(1).line\_number); END; / |