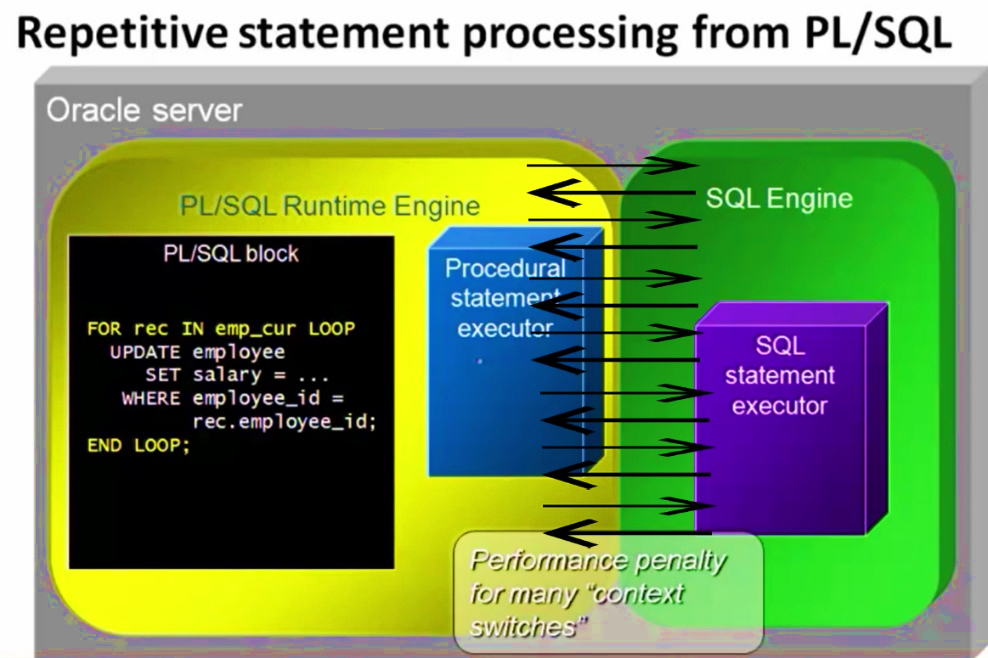
**Ejada Database Bulk Processing best practices**

**Version: 0**

# General Problem Definition:

Many PL/SQL blocks execute the same SQL statement repeatedly. — Retrieve data one row at a time. — Performs same DML operation for each row retrieved. But this will cause many context switches between PL/SQL Runtime Engine and SQL Engine for each item – row – due to using DML SQL for each item.

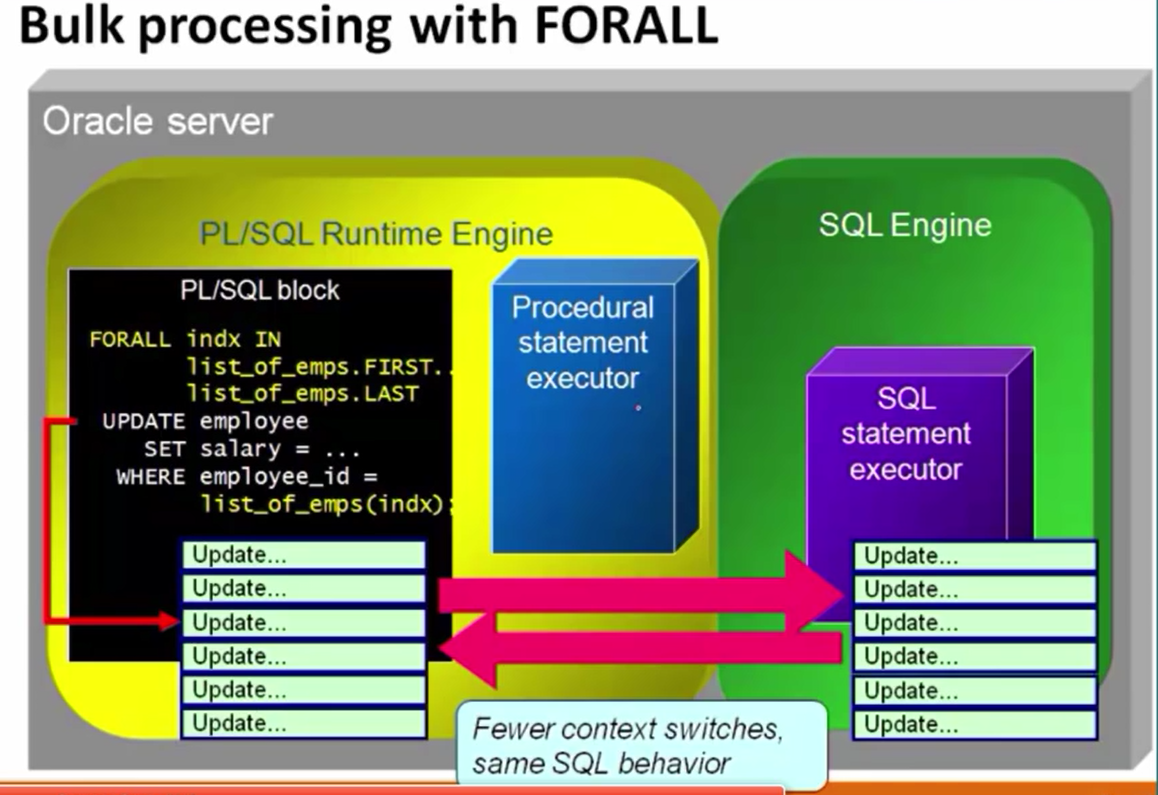


# General Solution Strategy:

Oracle PL/SQL provides the functionality of fetching or manipulating the records in bulk rather than fetching or manipulating them one-by-one. As a result we can reduce the number of context switches and improve performance. To do this, Oracle "bundles up" the requests for data (or to change data) and then passes them with a single context switch using FORALL and BULK COLLECT.

# FORALL

The FORALL allows to perform the DML operations on data in bulk. It is similar to that of FOR loop statement except in FOR loop things happen at the record-level whereas in FORALL there is no LOOP concept but the entire data present in the given range is processed at the same time.



## Features

* FORALL to be used with inserts, updates, deletes and merges.
* FORAL moves data from collections to tables.
* No need to declare loop\_variable for FORALL loop as it’s automatically declared.
* BEFORE and AFTER statement-level triggers only fire once per FORALL INSERT statements. Not for each INSERT statement passed to the SQL engine.
* Only one DML statement is allowed per FORALL.
* SQL%ROWCOUNT returns total number of rows modified by entire FORALL
* SQL%BULKCOUNT returns how number of rows modified by each statement.

## Syntax

FORALL <loop\_variable> IN <lower range> .. <higher range>

<DML operations>;

## Example

|  |  |
| --- | --- |
| Normal for loop FOR indx IN 1 .. num  LOOP  INSERT INTO parts  VALUES (collection (index) );  END LOOP; | ForALL loop FORALL indx IN 1 .. num  INSERT INTO parts  VALUES (collection (index) );  FORALL index IN 1..collection.COUNT  DELETE FROM employee  WHERE emp\_id = collection(index); |

# BULK COLLECT

With Oracle bulk collect, the PL/SQL engine tells the SQL engine to collect many rows at once and place them in a collection. During an Oracle bulk collect, the SQL engine retrieves all the rows and loads them into the collection and switches back to the PL/SQL engine. When rows are retrieved using Oracle bulk collect, they are retrieved with only two context switches.

## Features

* BULK COLLECT used with all kinds of queries: implicit, explicit, static and dynamic.
* BULK COLLECT moves data from tables into collections.
* NO\_DATA\_FOUND is not raised when no rows are fetched but the collection is empty.
* The "INTO" collections are filled sequentially from index value 1.
* Only integer-indexed collections with the same type of the data retrieved may be used.
* No need to initialize or extend nested tables and varrays.
* If you are certain that your table with never have more than N rows, use a VARRAY (N) and If that limit is exceeded, Oracle will raise an error
* If you do not know in advance how many rows you might retrieve, you should use an explicit cursor or Fetch BULK COLLECT with the LIMIT clause
* The LIMIT value is used to avoid excessive PGA memory consumption by retrieving data as patches with maximum length of LIMIT.
* LIMIT best performance to be passed as a parameter for flexibility.
* A limit of 100 seems like a good default value, As Setting it to 500 or 1000 doesn't seem to make much difference in performance.

## Syntax

SELECT \* BULK COLLECT INTO collection(s) FROM table;

OR

FETCH cur\_name BULK COLLECT INTO collection(s) LIMIT limit;

OR

EXECUTE IMMEDIATE query BULK COLLECT INTO collection(s);

## Example

DECLARE

TYPE employees\_aat IS TABLE OF employees%ROWTYPE;

1\_employees employees\_aat;

BEGIN

SELECT \*

BULK COLLECT INTO 1\_employees FROM employees;

FOR indx IN 1 .. l\_employees.COUNT

LOOP process\_employee (1\_employees(indx));

END LOOP;

END;

## Additional Hints

* You will need to break the habit of checking %NOTFOUND right after the fetch. As You might skip processing some of your data by mistake.

LOOP

FETCH my\_cursor BULK COLLECT INTO 1\_collection LIMIT 100;

EXIT WHEN my\_cursor%NOTFOUND;

* Instead, do one of the following:
  + At the end of the loop, check %NOTFOUND.
  + Right after fetch, exit when collection.COUNT = 0.
  + At end of loop, exit when collection.COUNT < limit.