Design Considerations for PL/SQL Code

Objectives

After completing this lesson, you should be able to do the following:

- Using the RETURNING Clause
- Using Bulk Binding
- Bulk Binding: FORALL Keyword
- Bulk Binding: BULK COLLECT Keyword
- Bulk Binding: Syntax and Keywords
- Using BULK COLLECT INTO with Queries
- Using BULK COLLECT INTO with Cursors

Using the RETURNING Clause

- Often, applications need information about the row affected by a SQL operation.
- e,g; To generate a report or to take a subsequent action.

- The INSERT, UPDATE, and DELETE statements can include a RETURNING clause, which returns column values from the affected row into PL/SQL variables or host variables.

- This eliminates the need to SELECT the row after an INSERT or UPDATE, or before a DELETE.

- As a result, fewer network round trips, less server CPU time, fewer cursors, and less server memory are required.

Using the RETURNING Clause

- Improves performance by returning column values with INSERT, UPDATE, and DELETE statements
- Eliminates the need for a SELECT statement

```
CREATE OR REPLACE PROCEDURE update_salary(p_emp_id
   NUMBER) IS
  v_name    employees.last_name%TYPE;
  v_new_sal employees.salary%TYPE;
BEGIN
  UPDATE employees
    SET salary = salary * 1.1
  WHERE employee_id = p_emp_id
  RETURNING last_name, salary INTO v_name, v_new_sal;
  DBMS_OUTPUT.PUT_LINE(v_name || ' new salary is ' ||
    v_new_sal);
END update_salary;
/
```

ORACLE


```
1 SET SERVEROUTPUT ON
2 /
3 SELECT last_name, salary
4 FROM employees
5 WHERE employee_id = 108;
6 /
7 EXECUTE update_salary(108)
```

```
LAST_NAME                 SALARY
------------------------- --------
Greenberg                  12008

anonymous block completed
Greenberg new salary is 13208.8
```

Using Bulk Binding

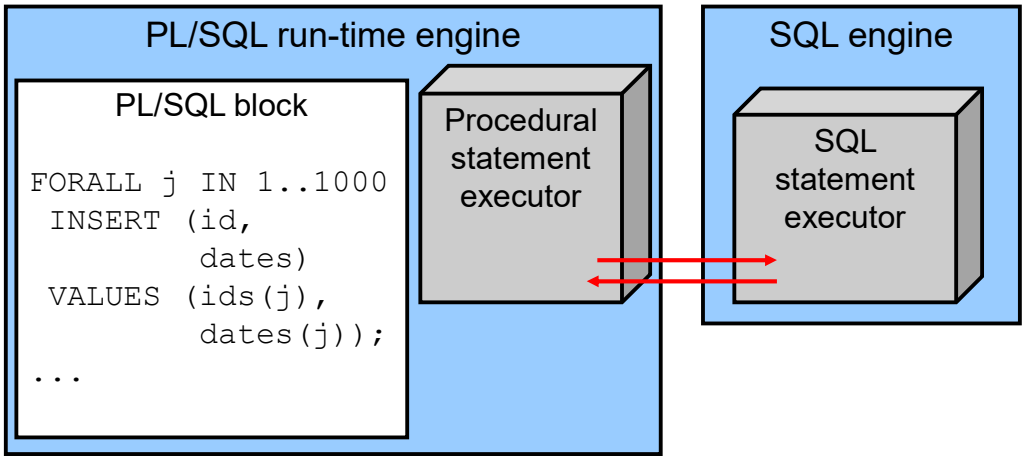
- The Oracle server uses two engines to run PL/SQL blocks and subprograms:
  - The PL/SQL run-time engine, which runs procedural statements but passes the SQL statements to the SQL engine
  - The SQL engine, which parses and executes the SQL statement and, in some cases, returns data to the PL/SQL engine
- During execution, every SQL statement causes a context switch between the two engines, which results in a performance penalty for excessive amounts of SQL processing.
- This is typical of applications that have a SQL statement in a loop that uses values from indexed collections.
- Collections include nested tables, varying arrays, index-by tables, and host arrays.

Using Bulk Binding

- Performance can be substantially improved by minimizing the number of context switches through the use of bulk binding.
- Bulk binding causes an entire collection to be bound in one call, a context switch, to the SQL engine.
- That is, a bulk bind process passes the entire collection of values back and forth between the two engines in a single context switch, compared with incurring a context switch for each collection element in an iteration of a loop.
- The more rows affected by a SQL statement, the greater the performance gain with bulk binding.

Using Bulk Binding

Binds whole arrays of values in a single operation, rather than using a loop to perform a `FETCH`, `INSERT`, `UPDATE`, and `DELETE` operation multiple times



Bulk Binding: FORALL Keyword

- Use bulk binds to improve the performance of:
  - DML statements that reference collection elements
    - `SELECT` statements that reference collection elements
  - Cursor `FOR` loops that reference collections and the `RETURNING INTO` clause
- The `FORALL` keyword instructs the PL/SQL engine to bulk bind input collections before sending them to the SQL engine.
- Although the `FORALL` statement contains an iteration scheme, it is not a `FOR` loop.

Bulk Binding: BULK COLLECT Keyword

- The `BULK COLLECT` keyword instructs the SQL engine to bulk bind output collections, before returning them to the PL/SQL engine.
- This enables you to bind locations into which SQL can return the retrieved values in bulk.
- Thus, you can use these keywords in the `SELECT INTO`, `FETCH INTO`, and `RETURNING INTO` clauses.

Bulk Binding: Syntax and Keywords

- The FORALL keyword instructs the *PL/SQL engine* to bulk bind input collections before sending them to the SQL engine.

```
FORALL index IN lower_bound .. upper_bound
  [SAVE EXCEPTIONS]
  sql_statement;
```

- The BULK COLLECT keyword instructs the *SQL engine* to bulk bind output collections before returning them to the PL/SQL engine.

```
... BULK COLLECT INTO
      collection_name[,collection_name] ...
```


The SAVE EXCEPTIONS keyword is optional. However, if some of the DML operations succeed and some fail, you would want to track or report on those that fail. Using the SAVE EXCEPTIONS keyword causes failed operations to be stored in a cursor attribute called %BULK_EXCEPTIONS, which is a collection of records indicating the bulk DML iteration number and corresponding error code.

Handling FORALL Exceptions with the %BULK_EXCEPTIONS Attribute

To manage exceptions and have the bulk bind complete despite errors, add the keywords SAVE EXCEPTIONS to your FORALL statement after the bounds, before the DML statement.

All exceptions raised during the execution are saved in the cursor attribute %BULK_EXCEPTIONS, which stores a collection of records. Each record has two fields:

%BULK_EXCEPTIONS(I).ERROR_INDEX holds the "iteration" of the FORALL statement during which the exception was raised and %BULK_EXCEPTIONS(i).ERROR_CODE holds the corresponding Oracle error code.

Values stored in %BULK_EXCEPTIONS refer to the most recently executed FORALL statement. Its subscripts range from 1 to %BULK_EXCEPTIONS.COUNT.

Note: For additional information about bulk binding and handling bulk-binding exceptions, refer to the *Oracle Database PL/SQL User's Guide and Reference.*

Bulk Binding FORALL: Example

```
CREATE PROCEDURE raise_salary(p_percent NUMBER) IS
  TYPE numlist_type IS TABLE OF NUMBER
    INDEX BY BINARY_INTEGER;
  v_id  numlist_type; -- collection
BEGIN
 v_id(1):= 100; v_id(2):= 102; v_id(3):= 104; v_id(4) := 110;
 -- bulk-bind the PL/SQL table
  FORALL i IN v_id.FIRST .. v_id.LAST
    UPDATE employees
      SET salary = (1 + p_percent/100) * salary
      WHERE employee_id = v_id(i);
END;
/
```

```
EXECUTE raise_salary(10)
```

```
anonymous block completed
```


Note: Before you can run the example in the slide, you must disable the `update_job_history` trigger as follows:

```
ALTER TRIGGER update_job_history DISABLE;
```

In the example in the slide, the PL/SQL block increases the salary for employees with IDs 100, 102, 104, or 110. It uses the `FORALL` keyword to bulk bind the collection. Without bulk binding, the PL/SQL block would have sent a SQL statement to the SQL engine for each employee record that is updated. If there are many employee records to update, the large number of context switches between the PL/SQL engine and the SQL engine can affect performance. However, the `FORALL` keyword bulk binds the collection to improve performance.

Note: A looping construct is no longer required when using this feature.

An Additional Cursor Attribute for DML Operations

Another cursor attribute added to support bulk operations is `%BULK_ROWCOUNT`. The `%BULK_ROWCOUNT` attribute is a composite structure designed for use with the `FORALL` statement. This attribute acts like an index-by table. Its *i*th element stores the number of rows processed by the *i*th execution of an `UPDATE` or `DELETE` statement. If the *i*th execution affects no rows, then `%BULK_ROWCOUNT(i)` returns zero.

Here is an example:

```
CREATE TABLE num_table (n NUMBER);
DECLARE
  TYPE num_list_type IS TABLE OF NUMBER
    INDEX BY BINARY_INTEGER;
  v_nums num_list_type;
BEGIN
 v_nums(1) := 1;
 v_nums(2) := 3;
 v_nums(3) := 5;
 v_nums(4) := 7;
 v_nums(5) := 11;
  FORALL i IN v_nums.FIRST .. v_nums.LAST
    INSERT INTO v_num_table (n) VALUES (v_nums(i));
  FOR i IN v_nums.FIRST .. v_nums.LAST
  LOOP
    dbms_output.put_line('Inserted ' ||
      SQL%BULK_ROWCOUNT(i) || ' row(s)'
      || ' on iteration ' || i);
  END LOOP;
END;
/
DROP TABLE num_table;
```

The following results are produced by this example:

```
table NUM_TABLE created.
anonymous block completed
Inserted 1 row(s) on iteration 1
Inserted 1 row(s) on iteration 2
Inserted 1 row(s) on iteration 3
Inserted 1 row(s) on iteration 4
Inserted 1 row(s) on iteration 5

table NUM_TABLE dropped.
```

Using BULK COLLECT INTO with Queries

The SELECT statement supports the BULK COLLECT INTO syntax.

```
CREATE PROCEDURE get_departments(p_loc NUMBER) IS
  TYPE dept_tab_type IS
    TABLE OF departments%ROWTYPE;
  v_depts dept_tab_type;
BEGIN
  SELECT * BULK COLLECT INTO v_depts
  FROM departments
  WHERE location_id = p_loc;
  FOR i IN 1 .. v_depts.COUNT LOOP
    DBMS_OUTPUT.PUT_LINE(v_depts(i).department_id
     ||' '|| v_depts(i).department_name);
  END LOOP;
END;
```


Starting with Oracle Database 10*g*, when using a SELECT statement in PL/SQL, you can use the bulk collection syntax shown in the example in the slide. Thus, you can quickly obtain a set of rows without using a cursor mechanism.

The example reads all the department rows for a specified region into a PL/SQL table, whose contents are displayed with the FOR loop that follows the SELECT statement.

Using BULK COLLECT INTO with Cursors

The FETCH statement has been enhanced to support the BULK COLLECT INTO syntax.

```
CREATE OR REPLACE PROCEDURE get_departments(p_loc NUMBER) IS
  CURSOR cur_dept IS
    SELECT * FROM departments
    WHERE location_id = p_loc;
  TYPE dept_tab_type IS TABLE OF cur_dept%ROWTYPE;
  v_depts dept_tab_type;
BEGIN
  OPEN cur_dept;
  FETCH cur_dept BULK COLLECT INTO v_depts;
  CLOSE cur_dept;
 FOR i IN 1 .. v_depts.COUNT LOOP
    DBMS_OUTPUT.PUT_LINE(v_depts(i).department_id
     ||' '|| v_depts(i).department_name);
  END LOOP;
END;
```

Quiz

The NOCOPY hint allows the PL/SQL compiler to pass OUT and IN OUT parameters by reference rather than by value. This enhances performance by reducing overhead when passing parameters.

a. True
b. False

Answer: a

PL/SQL subprograms support three parameter-passing modes: IN, OUT, and IN OUT.

By default:

- The IN parameter is passed by reference. A pointer to the IN actual parameter is passed to the corresponding formal parameter. So, both the parameters reference the same memory location, which holds the value of the actual parameter.
- The OUT and IN OUT parameters are passed by value. The value of the OUT or IN OUT actual parameter is copied into the corresponding formal parameter. Then, if the subprogram exits normally, the values assigned to the OUT and IN OUT formal parameters are copied into the corresponding actual parameters.

Copying parameters that represent large data structures (such as collections, records, and instances of object types) with OUT and IN OUT parameters slows down execution and uses up memory. To prevent this overhead, you can specify the NOCOPY hint, which enables the PL/SQL compiler to pass OUT and IN OUT parameters by reference.

Summary

In this lesson, you should have learned how to:

- Using the RETURNING Clause
- Using Bulk Binding
- Bulk Binding: FORALL Keyword
- Bulk Binding: BULK COLLECT Keyword
- Bulk Binding: Syntax and Keywords
- Using BULK COLLECT INTO with Queries
- Using BULK COLLECT INTO with Cursors