Exploring PL/SQL New Features & Best Practices for Database Developers

Ami Aharonovich
ilOUG, DBAces

Keywords:

PL/SQL
Best Practices
PL/SQL Performance
11g New Features
12c New Features

Introduction

Oracle Database 11g and Oracle Database 12c brings many new features and enhancements to PL/SQL. These will help you to improve your applications and database performance, enhance applications functionality, provide better security for your applications and increase your productivity as a database developer.

This session will present best practices for implementing Oracle database 11g new PL/SQL programming language features and enhancements that can be used to improve programming functionality, performance and usability.

Participants will learn about new performance related features, new trigger options, PL/SQL function result cache, bulk binding enhancements, subprogram inlining, new security features and more.

The session will also include an introduction to a few PL/SQL key new features included in Oracle Database 12c version.

Oracle SQL Developer – Overview

Oracle SQL Developer is a free and fully supported graphical integrated development environment that simplifies the development and management of the Oracle database. Oracle SQL Developer offers a complete end-to-end development of your PL/SQL applications, worksheet for running queries and scripts, DBA console for managing the database, enhanced reports interface, data modelling solutions and a migration platform for moving 3rd party databases into Oracle database.

In this first part of the presentation we will briefly discuss a few key features in Oracle SQL Developer and will also demonstrate a few common and important features that can be used on a daily basis.

Parsing Time is Important

Here I would like to present a simple demo, where we insert 100,000 new rows into a table, using two different anonymous blocks. The first one will use dynamic SQL (with EXECUTE IMMEDIATE) without using a bind variable while the second one will basically do exactly the same just with a small and significant difference – it will use a bind variable. Here is the code to be executed:

```
BEGIN
FOR i IN 1..100000 LOOP
EXECUTE IMMEDIATE 'INSERT INTO t (x,y) VALUES ('||i||',''A'')';
END LOOP;
END;
```

The above version is without the usage of bind variable and it will take about 40 seconds to complete.

```
BEGIN
FOR i IN 1..100000 LOOP
EXECUTE IMMEDIATE 'INSERT INTO t (x,y) VALUES (:i,''A'')' USING i;
END LOOP;
END;
```

The above version is with the usage of bind variable and it will take about 8 seconds to complete.

Now, we will query the data dictionary views to get the amount of memory which is allocated for each one of the insert statements. We will see the number of cursors that are still kept in cache and the total amount of memory allocated for those statements. Here is the query that will present the above:

```
SELECT SUBSTR(sql_text,11,8) "Bind", COUNT(*),
       ROUND(SUM(sharable_mem)/1024) "Memory KB"
FROM   v$sqlarea
WHERE  sql_text LIKE 'INSERT%INTO t (x,y)%'
GROUP BY SUBSTR(sql_text,11,8);
```

The above query will show a significant difference between the two anonymous blocks. The first one, without the bind variable, will have thousands of versions kept in the memory consuming a few hundreds of MB from the total amount of memory allocated for the shared pool. The second one, using the bind variable, will have a single version kept in memory and it will allocate only 14 KB of memory. That is a huge difference and of course a significant benefit for the usage of binds variables.

We will also discuss the bind variable peeking feature, the fact that the Oracle optimizer peeks at the binds value in order to determine how to optimize the query, when the query is first hard parsed. In addition, we will include also a short discussion on the 11g new feature called Adaptive Cursor Sharing, which provide an intelligent cursor sharing mechanism for statements that use bind variables and allows the optimizer to generate a set of plans that are optimal for different sets of bind values.

Using Bulk Binding

PL/SQL is made up of two types of code: procedural and SQL, each one processed by a different engine. There is an overhead associated with switching from procedural code to SQL and back again because of the switching between the PL/SQL and SQL engines. In relatively small operations these switches are barely perceivable, but they can become a problem during large looping operations where the delay is multiplied many times. To prevent this problem, Oracle provides a mechanism that in a single operation binds DML statements to whole collections, reducing the number of context switches. This mechanism is known as bulk binding and is the subject of this chapter.

We will discuss the different bulk features, including BULK COLLECT for SELECT statements and FORALL which is used for DML statements and will also run a few examples to compare between running a PL/SQL procedure with and without using the bulk features.

Following is a PL/SQL anonymous block to present the usage of bulk biding for doing a large insert operation in bulk, on 1 million rows, using a PL/SQL index by table, doing a single context switch

```
DECLARE
TYPE part_num_type IS TABLE OF parts.part_num%TYPE
INDEX BY PLS_INTEGER;
TYPE part_name_type IS TABLE OF parts.part_name%TYPE
INDEX BY PLS_INTEGER;
v_part_num    part_num_type;
v_part_name   part_name_type;
BEGIN
FOR i IN 1..1000000 LOOP
v_part_num(i) := i;
v_part_name(i):= 'Part '||i;
END LOOP;
FORALL i IN v_part_num.FIRST..v_part_num.LAST
INSERT INTO parts VALUES (v_part_num(i),v_part_name(i));
END;
/
```

Oracle 11g PL/SQL Function Result Cache

In the past, if you called a PL/SQL function 1,000 times and each function call consumed 1 second, the 1,000 calls would take 1,000 seconds. With this 11g new Function Results Cache feature, depending on the inputs to the function and whether the data underlying the function changes, 1,000 function calls could take about 1 second, total. The PL/SQL Function Result Cache allows storing the results of PL/SQL functions in the SGA (in the shared pool). This new feature provides the ability to mark a PL/SQL function to indicate that its result should be cached to allow lookup, rather than recalculation, when the same parameter values are called. This is done transparently using the input parameters as the lookup key and it is instance wide – meaning that all distinct sessions invoking the function can benefit from this new feature.

Subprogram Inlining

Every call to a procedure or function causes a slight, but measurable, performance overhead, which is especially noticeable when the subprogram is called within a loop. Automatic subprogram inlining can reduce the overheads associated with calling subprograms, whilst leaving your original source code in its normal modular state. This is done by replacing the subprogram calls with a copy of the code in the subprogram at compile time.

We will discuss this interesting feature and the advantages of using it and will also see an example where subprogram inlining can be either manually or automatically implemented in a PL/SQL block and will compare the performance between the two options, using and not using this new feature.

Following is the PL/SQL anonymous block example which presents the usage of subprogram inlining:

```
DECLARE
      l_loops  NUMBER := 10000000;
      l_return NUMBER;
      FUNCTION add_numbers (p_1 IN NUMBER, p_2 IN NUMBER)
      RETURN NUMBER IS
      BEGIN
               RETURN p_1 + p_2;
      END add_numbers;
BEGIN
      FOR i IN 1..l_loops LOOP
               PRAGMA INLINE (add_numbers, 'YES');
               l_return := add_numbers(1, i);
      END LOOP;
END;
/
```

Finer Grained Dependencies

In previous releases, metadata recorded mutual dependencies between objects with the granularity of the whole object. This means that dependent objects were sometimes invalidated when there was no logical requirement to do so. Oracle Database 11*g* records dependency metatdata at a finer level of granularity, meaning that it is reducing the consequential invalidation of dependent objects in response to changes in the objects they depend upon, and so application availability is increased. The benefit is felt both in the development environment and when a live application is parsed or upgraded.

We will see an example where we benefit from this new and improved database behavior and will compare between Oracle database 10g and Oracle database 11g considering objects dependencies.

Oracle Database 12c New Features Overview

Towards the ending of this presentation, we will briefly describe a few key new features presented in Oracle Database 12c, including:

1) Using invisible columns – describing the usage of this feature and describing an example showing how this feature can be used for changing the order of columns in an existing table structure

2) Invoking PL/SQL from SQL – describing the usage of WITH clause with a PL/SQL function and how we can invoke this PL/SQL procedural code from within the SQL statement

3) Using new improved default values – describing how we can use new defaults for sequences, defaults when nulls are inserted and the new identity type

4) Enhanced statistics capabilities – describing new enhanced statistics capabilities for gathering statistics during loads, and gathering session private statistics for global temporary tables

Following is an example for using the new invisible column feature presented in Oracle Database 12c and how it can be used to change the order of columns in an existing table structure:

```
CREATE TABLE demo
(col1 NUMBER, col2 NUMBER, col3 NUMBER INVISIBLE);
INSERT INTO demo VALUES (1,2);
INSERT INTO demo (col1,col2,col3) VALUES (1,2,3);
SELECT * FROM demo;
SELECT col1,col2,col3 FROM demo;
ALTER TABLE demo MODIFY (col2 INVISIBLE);
ALTER TABLE demo MODIFY (col3 VISIBLE);
```

```
SELECT column_name,column_id
FROM   user_tab_columns
WHERE  table_name='DEMO';
```

General PL/SQL Best Practices and Guidelines

For the closure of this presentation, we will briefly discuss a few general PL/SQL best practices, tips and guidelines:

1) Avoiding the usage of procedural code when SQL code can be better

2) Use bulk binds to reduce context switches between the PL/SQL engine and the SQL engine

3) Prefer using External Table instead of using UTL_FILE to read text files

4) Prefer using SQL MERGE statements instead of using PL/SQL merge operations

5) Use DML error handling (DBMS_ERRLOG) for trapping failures in large DML operations rather than coding PL/SQL for achieving the same results

6) Beware of implicit data conversion and its impact on SQL performance

7) Modularize your code, limit the number of lines of code between a BEGIN and END, use packaged programs to keep smaller executable sections, use local procedure and functions to hide logic and use function interface to hide formulas and business rules

8) Write readable code, use UPPER and lower case, use indentation, avoid using hard-coded literals, use anchored declarations when possible, use cursor for loop

This presentation will include real life examples and live demonstrations using Oracle Database 11g and Oracle Database 12c with SQL*Plus and SQL Developer. Slides will also include some references for additional readings and recommended articles for Oracle Database 11g/12c New Features Guide, Oracle Database PL/SQL Language Reference 12c, PL/SQL performance enhancements and writing efficient PL/SQL code and a few Oracle Database 12c new features documents.

Contact address:

Ami Aharonovich
ilOUG, DBAces
9 Shaul Hameleh St.
55654, Kiryat Ono

Phone:	+972-524-377737
Fax:	+972-77-3373770
Email	Ami@DBAces.co.il
Internet:	www.DBAces.co.il