Oracle PLSQL Coding Guidelines

1. Overview

1.1 Scope

This document defines the standards and guidelines that will be used by developers when submitting PL/SQL components. These guidelines will also be used by reviewers to determine whether a component submission meets TopCoder standards.

2. Standards

2.1 Packages and Procedures

All PL/SQL functions and procedures will be implemented as part of a package. The package names used will be specified in the component specification.

2.2 Unit Testing

All submitted components will include unit tests written with the 'utPLSQL' package. The package is available at the URL given in the 'references' section of this document.

2.3 Source Files

Package headers and bodies will be declared in separate files. Package header file names will use the 'pkg' extension. Package body file names will use the '.pkb' extension.

Trigger code will be declared in a single file with the extension '.trg'.

2.3.1 Header Comments

Package header source files will contain a package level comment which includes the following information: FileName, Component Name, Description, Package Name, Designer Name, Developer Name, Version, Date and Copyright statement. The format below will be followed exactly.

```
/*****************************************************************
/*
/* Filename: TopCoder_Unit_Test.pkg
/* Component: Unit_Test_Framework
/* Package: Unit_Test_Frmwrk
/* Designer: TCSDesigner
/* Developer: TCSDeveloper
/* Version: 1.0
/* Copyright (c) 2006, TopCoder, Inc.  All rights reserved.
/*
/* Description: Description of PL/SQL package...
/*
/*****************************************************************
```

These comments will be used at a later time by an automated comment generator (possibly RoboDoc) to generate HTML package documentation.

2.3.2 Function/Procedure Comments

The package header source file will document each function in the following format:

```
/*****************************************************************
/**
/** Function: get_employee_ssn
```

```
/** In: p_employee_id – the id of the employee to search for
/** Returns: the Social Security Number for the employee
/**
/*********************************************************************
```

The package header source file will document each procedure in the following format:

```
/*********************************************************************
/**
/** Procedure:   ins_employee
/** Out: p_employee_id – the id of the newly created employee.
/** In: p_ssn – the Social Security Number of the employee to
/**               insert.
/** In: p_name – the name of the employee to insert.
/**
/*********************************************************************
```

The possible parameter types are 'In', 'Out' and 'InOut'. These comments will be used at a later time by an automated comment generator (possibly RoboDoc) to generate HTML package documentation.

2.3.3 Maximum Line Length

No single line of code in a PL/SQL component will exceed 120 characters in length.

2.3.4 Indentation

Indentation will be 0 spaces for the outermost block and 3 spaces from the indentation level of each enclosing block.

2.3.5 Alignment

Comma separated lists will be 'stacked' and aligned as shown below:

```
SELECT SUM(A)
     , B
     , C
  FROM TABLE1
 WHERE B = 5
GROUP BY B
     , C
```

2.3.6 Conditional Blocks

In a conditional, the 'THEN' keyword will be placed on the line below the 'IF' but aligned with it. 'ELSEIF' keywords will also be aligned with the 'IF'.

Example:

```
IF l_total > lc_max
THEN
   l_new_max := true;
ELSIF l_total = lc_max
THEN
   l_new_max := false;
END IF;
```

2.4 Reserved Words

SQL and PL/SQL reserved words (SELECT, INSERT, PACKAGE, FUNCTION, etc) will be capitalized.

2.5 Variable Names

Variable names will be all lower case, with individual words separated by an underscore. The following standard prefixes will be used:

Prefix	Usage Context
p_	Function and procedure parameters
l_	Function and procedure local variables
g_	Package global variables
lc_	Function and procedure local constants
gc_	Package global constants

2.6 Variable Types

2.6.1 Row and Column Types

When there is a direct correlation between a variable and a table column, the %TYPE or %ROWTYPE will be used.

Example:

```
DECLARE
      l_wins team.wins%TYPE;
```

2.6.2 Subtypes

When there's no direct correlation between a variable and table column variable restrictions will not be hard-coded. The developer will use 'SUBTYPE' to standardize data types.

Example:

```
CREATE OR REPLACE PACKAGE team_data
      SUBTYPE total_win_count_t IS INTEGER(10);
…
      DECLARE
            l_win_count team_data.total_win_count_t;
```

2.7 Variable Initialization

Variables will only be initialized in the 'DECLARE' section when that initialization doesn't require a function call or complex logic. When the variable must be initialized via a function call or complex logic, it will be done in the executable section of the procedure or function.

2.8 Function and Procedure Naming

Functions and procedures will be names using the following guidelines. Functions and procedures that don't fall into one of these usage contexts will be named descriptively at the developer's discretion.

Prefix	Usage Context
ins_	Procedures whose primary purpose is to insert data
upd_	Procedures whose primary purpose is to update data
del_	Procedures whose primary purpose is to delete data
get_	Functions whose primary purpose is to retrieve data
chk_	Functions which return a Boolean value

2.9 Encapsulation

2.9.1 DML Statements

Insert and Update statements will be encapsulated in a single package procedure or function. This will allow all inserts and updates to use identical SQL which will decrease statement parsing.

2.9.2 Package Data

Non-constant package level data will be declared in the package body (privately), not in the package specification. 'get_' and (optionally) 'set_' procedures will be included to modify this data from outside the package when necessary. The package specification may include constant, type and cursor definitions.

2.10 Control Structures

2.10.1 Conditionals

The 'ELSIF/ELSE' construct will be used in preference to multiple or nested conditionals.

Example:
These statements:

```
IF l_count_1 = l_count_2
THEN
  // handle case 1
END IF;

IF l_count_1 > l_count_2
THEN
  // handle case 2
ELSE
  IF l_count_2 > l_count1
  THEN
    // handle case 3
  END IF;
END IF;
```

Should be written as:

```
IF l_count_1 = l_count_2
THEN
  // handle case 1
ELSIF l_count_1 > l_count_2
THEN
  // handle case 2
ELSIF l_count_2 > l_count1
THEN
  // handle case 3
END IF;
```

2.10.2 Loop Exits

It's considered bad practice to exit from the middle of a loop. Each loop should include only a single exit point.

2.10.3 GOTO Statement

PL/SQL's GOTO statement will not be used.

2.11 Exceptions

2.11.1 Raising Exceptions

PL/SQL components will not raise exceptions directly. The component will use a single procedure to encapsulate exception raising logic. Exceptions will be used to indicate errors, not as a normal method for branching control.

2.11.2 Custom Exceptions

Custom exceptions will be declared in the package specification. Custom exceptions will not hide an underlying exception. For example, the developer should not catch a system-created exception and then raise an application exception instead.

2.11.3 Exception Handling

Exception handling code will catch specific exceptions and will only use the 'WHEN OTHERS' construct in the outermost code. Every effort should be made to preserve the root cause of exceptions to the client code.

2.12 Cursors

2.12.1 Declarations

Multi-row cursors are often useful to users outside the package and should be included in the package specification at the designer's discretion.

2.12.2 Cursor For Loop

A cursor for loop will not be used to retrieve a single row. Cursor for loops will only be used when every row in the cursor will be processed. If the loop exits before the all rows are read from the cursor, then the cursor will not be properly closed.

2.12.3 Fetching

Cursors will be fetched into cursor records, not into multiple variables.

2.12.4 Select For Update

Select for update will be used when one or more of the selected rows will be updated.

2.13 SQL and Dynamic SQL

2.13.1 Bind Variables

Bind variables (sometimes called Prepared Statements or Parameter Markers) will be used whenever possible. All non-dynamic SQL statements will use bind variables.

2.13.2 Context Switching

For performance reasons, switching contexts between SQL and PL/SQL execution should be minimized. Single-row queries will be placed in their own function.

2.14 Functions

2.14.1 Boolean Functions

The only valid values that may be returned from Boolean functions are 'true' and 'false'. NULL should never be returned from a Boolean function. Boolean functions may raise exceptions.

2.14.2 OUT Parameters

Functions should not contain 'OUT' parameters.

2.15 Transaction Control

Transactions will generally be managed by the calling component. Test code may contain 'commit' and 'rollback' statements, but the PL/SQL component code will not.

3. References

utPLSQL - http://www.oreillynet.com/pub/a/oreilly/oracle/utplsql/news/news.html

Oracle PL/SQL Best Practices, Steven Feuerstein, 2001