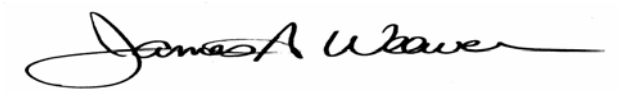
COMMONWEALTH OF PENNSYLVANIA
DEPARTMENT OF PUBLIC WELFARE

INFORMATION TECHNOLOGY STANDARD

Name Of Standard: **Oracle PL/SQL Coding and Naming**	Number: **STD-DMS001**
Domain: **Data**	Category: **Oracle PL/SQL Coding**
Date Issued: **08/14/2003**	Issued By Direction Of:  James Weaver, Dir of Div of Tech Engineering
Date Revised: **10/20/2009**	

Abstract:

The standards outlined in this document are the result of collaborations among members of the DBA Technical team.

The subgroup focused on developing Oracle's Procedural Language Structured Query Language (PL/SQL) coding standards to be used in situations where there is information being shared among the Department of Public Welfare (DPW) and their business partners. A business partner can be identified as an external business entity (e.g., the Social Security Administration, IRS, etc.) or another agency or department within the Commonwealth (e.g., BCSE), or even between applications within the Department (e.g., HCSIS, CCMIS, PACSES).

The remainder of this document outlines the general standards that have been proposed and approved by the DBA Technical team. The sections that follow include:

- Oracle PL/SQL Overview – defines what PL/SQL is and identifies benefits to using the language
- Oracle PL/SQL Basic Components – basic PL/SQL terminology and standards are defined and explained
- DPW PL/SQL Standards – documents the specific PL/SQL standards that have been developed as a result of this initiative
- Appendix I – provides a glossary of PL/SQL terms used throughout this document

General:

This document contains DPW naming and coding standards for Oracle PL/SQL followed by the Bureau of Information Systems (BIS) in supporting the Department of Public Welfare business systems. Oracle PL/SQL Coding and Naming Standards supplement the standards contained in the Governor's Office of

Administration/ Office of Information Technology (OA/OIT) Information Technology Bulletin **ITB-INF001** Database Management Systems and **ITB-APP001** Business Solutions Center of Excellence (BSCoE).

Coding standards are methods that developers follow when developing, editing, or maintaining program code. Uniform programming style, developer understanding, readability, enhanced performance, reusable code, reduced application development time, and ease of maintenance are the results of developers following coding standards.

Applicability

These standards apply to all Oracle DPW application databases with the exception of the ITASCA and Oracle APEX databases and do not apply to Commercial Off-The-Shelf (COTS) databases.

These standards apply to all new development and to existing stored procedures and packages when there is a significant change. Random code reviews using a standard checklist will be completed by the Database Design Unit to monitor adherence to the standards developed by the joint DBA Technical team.

Coding Standard:

Oracle PL/SQL is the standard coding language to be used for development except in cases when using the ANSI SQL standard is recommended. Once such exception is Oracle's recommendation to use the ANSI join over the Oracle join operator.

Oracle PL/SQL Overview

PL/SQL Description

PL/SQL is Oracle's "procedural extension to Structured Query Language (SQL)." In simple terms, PL/SQL is a computer language which enhances and extends SQL by enabling the use of conditional statements. Being a language extension, PL/SQL uses a set of features to enhance an existing language. PL/SQL is not a version of SQL; it is a complete computer language with its own syntax, rules and compiler. It replaces those procedural ingredients that SQL took out: sequential, conditional, and iterative statements, variables, named programs, and more. The PL/SQL language relates to SQL, but writes programs as an ordered series of statements.

PL/SQL is most commonly used to write data-centric programs in order to manipulate data in an Oracle database.

PL/SQL Usage Benefits

In comparison to using other languages – including Java – for system development, PL/SQL has many advantages:

- PL/SQL is more concise. In order to include SQL statements no superfluous code is needed. This means using fewer lines of code in comparison to other languages.
- PL/SQL functions are called directly from SQL which makes SQL statements shorter and more manageable.
- PL/SQL can run without any human intervention. Other events occurring in a database can automatically trigger PL/SQL code to execute.

- PL/SQL is data centric and tightly integrated into the Oracle database.
- Data manipulation is slightly faster in PL/SQL than in other languages.

PL/SQL Standardization Benefits

- Fewer decisions. Standards provide a starting block for coding development and remove the need to justify, argue and defend each decision during development process.
- Easy maintenance. Maintenance programming becomes easier since the code can be more efficiently read and modified if needed.
- Simplified training. By using standards it is easy to set up training programs, whether it is a hands-on-session or computer based training (CBT). New developers can be quickly trained to conform to the standards.
- Legacy automatic formatting. By using well-defined standards, it is possible to use an automatic formatting program to conform legacy code to updated standards.

Oracle PL/SQL Basic Components

The standards that follow in this section are continually evolving.

Block Structure

PL/SQL is structured into blocks and can use conditional statements, loops and branches to control the program flow. Variables can be scoped so that they are only visible within the block where they are defined. PL/SQL blocks come in three types: anonymous procedure, named procedure and named function. All of these block types share most PL/SQL features. The rules of block structure are:

- Every unit of PL/SQL must constitute a block. As a minimum there must be the delimiting words BEGIN and END around the executable statements.
- SELECT statements within PL/SQL blocks are embedded SQL (an ANSI category). As such they must return one row only. SELECT statements that return no rows or more than one row will generate an error. If you want to deal with groups of rows you must place the returned data into a cursor. The INTO clause is mandatory for SELECT statements within PL/SQL blocks (which are not within a cursor definition); you must store the returned values from a SELECT.
- If PL/SQL variables or objects are defined for use in a block then you must also have a DECLARE section.
- In an EXCEPTION section the statements within the section are only processed if the condition to which they refer occurs. Block execution is terminated after an exception handling routine is executed.
- PL/SQL blocks may be nested, nesting can occur wherever an executable statement could be placed (including the EXCEPTION section).

The basic format of Block Structure is as follows:

```
DECLARE
        (Definition of any variables or objects that are used within the declared
          block)
BEGIN
        (Statements that make up the block)
EXCEPTION
        (All exception handlers)
END;
        (End of block marker)
```

Error Handling

When errors occur in PL/SQL code, they are of two types:

- Internal exceptions that occur when Oracle code is violated that are handled automatically.
- User-defined or application specific exceptions that must be handled with a RAISE statement.

All stored procedures must contain an error-handling/exception section.

Build an error handling package that will be reused by all developers in the error-handling part of the PL/SQL block.

The function SQLERRM returns the error message associated with its error-number argument. If the argument is omitted, it returns the error message associated with the current value of SQLCODE. SQLERRM with no argument is useful only in an exception handler. Outside a handler, SQLERRM with no argument always returns the normal, successful completion message. For internal exceptions, SQLERRM returns the message associated with the Oracle error that occurred. The message begins with the Oracle error code.

SQLERRM is especially useful in the OTHERS exception handler, where it lets you identify which internal exception was raised. The error number passed to SQLERRM should be negative. Passing a zero to SQLERRM always returns the ORA-0000: normal, successful completion message. Passing a positive number to SQLERRM always returns the User-Defined Exception message unless you pass +100, in which case SQLERRM returns the ORA-01403: no data found message.

SQLERRM cannot be used directly in a SQL statement. Assign the value of SQLERRM to a local variable first, as shown in:

```
DECLARE
  name employees.last_name%TYPE;
  v_code NUMBER;
  v_errm VARCHAR2(64);
BEGIN
  SELECT last_name INTO name FROM employees WHERE employee_id = 1000;
  EXCEPTION
    WHEN OTHERS THEN
      v_code := SQLCODE;
      v_errm := SUBSTR(SQLERRM, 1 , 64);
      DBMS_OUTPUT.PUT_LINE('The error code is ' || v_code || '- ' || v_errm);
END;
/
```

DBMS_UTILITY.FORMAT_ERROR_STACK Function

This function formats the current error stack and can be used in exception handlers to look at the full error stack.

Syntax
DBMS_UTILITY.FORMAT_ERROR_STACK
  RETURN VARCHAR2;

Return Values

This returns the error stack, up to 2000 bytes

Stored Procedures

Stored procedures are an essential benefit of using PL/SQL. A stored procedure is a program that resides and executes inside the database server. They function easily on multi-tiered arrangements; the middle tier runs the application logic on some convenient platform using some convenient language. Some significant benefits of using stored procedures are as follows:

- By relying on stored procedures, there are fewer "moving" parts in the overall system; therefore, having fewer things to break. Controlling a development effort with client, middle-tier, and server-tier components requires not only inspired and skillful managers but also, in many cases, a great deal of luck. In addition, as time goes on, the evolution of a system with fewer components is likely to be simpler and less expensive.
- Stored procedures provide greater assurance of data integrity. It is easier not having to secure both the database and a middle tier. The term "secure" here encompasses both privileges and business rules.
- Stored procedures can potentially yield greater performance. PL/SQL's variables store data in the same internal binary format as the database. This means that when retrieving a numeric value from the database, it does not have to be converted from one set of bits in the database to a different set of bits in the program; just make an exact copy of the bits. After a magnitude of occurrences, stored procedures make a large impact on database performance.
- Stored procedures can facilitate greater productivity when writing applications with products that assume the presence of tables in the database. In Oracle, it is possible to write stored procedures that allow other programs to insert, update, and delete through database views.

Functions

A function is one of the basic units (procedures, functions, and anonymous blocks) or logical blocks that make up a PL/SQL program. These logical blocks, which can be nested inside one another.

Oracle® Database PL/SQL User's Guide and Reference 10g Release 2 (10.2)

Packages

A package is a schema object that groups logically related PL/SQL types, variables, and subprograms. Packages usually have two parts, a specification (spec) and a body; sometimes the body is unnecessary. The specification is the interface to the package. It declares the types, variables, constants, exceptions, cursors, and subprograms that can be referenced from outside the package. The body defines the queries for the cursors and the code for the subprograms.

Oracle® Database PL/SQL User's Guide and Reference 10g Release 2 (10.2)

Triggers

A trigger is a special kind of PL/SQL anonymous block. You can define triggers to fire before or after SQL statements, either on a statement level or for each row that is affected. You can also define INSTEAD OF triggers or system triggers (triggers on DATABASE and SCHEMA). Triggers are procedures that are stored in the database and implicitly run, or fired, when something happens.

Traditionally, triggers supported the execution of a PL/SQL block when an INSERT, UPDATE, or DELETE occurred on a table or view. Starting with Oracle8i, triggers support most system and other data events on DATABASE and SCHEMA.

A trigger is either a stored PL/SQL block or a PL/SQL, C, or Java procedure associated with a table, view, schema, or the database itself. Oracle automatically executes a trigger when a specified event takes place, which may be in the form of a system event or a DML statement being issued against the table.

Triggers can be:

- DML triggers on tables.
- INSTEAD OF triggers on views.
- System triggers on DATABASE or SCHEMA: With DATABASE, triggers fire for each event for all users; with SCHEMA, triggers fire for each event for that specific user.

Use the following standards when designing your triggers:

- Use triggers to guarantee that when a specific operation is performed, related actions are performed.
- Do not define triggers that duplicate features already built into Oracle. For example, do not define triggers to reject bad data if the same checking can be done through declarative integrity constraints.
- Limit the size of triggers. If the logic for the trigger requires much more than 60 lines of PL/SQL code, it is better to include most of the code in a stored procedure and call the procedure from the trigger.
- Use triggers only for centralized, global operations that should be fired for the triggering statement, regardless of which user or database application issues the statement.
- Do not create recursive triggers. For example, creating an AFTER UPDATE statement trigger on the Emp_tab table that itself issues an UPDATE statement on Emp_tab, causes the trigger to fire recursively until it has run out of memory.
- Use triggers on DATABASE judiciously. They are executed for every user every time the event occurs on which the trigger is created.

*Information gathered through online sources: O'Reilly's Learning Oracle PL/SQL and http://www.ilook.fsnet.co.uk/index.htm.

DPW ORACLE PL/SQL Standards

The prior section of this document included general standards for developers. The standards in this section are DPW naming and formatting conventions with specific examples provided.

Oracle's SQL Plus is the standard for compiling stored procedures.

Naming Conventions

All names are to be descriptive and meaningful. The program object should be named in such a way that it represents the business logic. A procedure used to generate reject letters for a set of records would be named USP_GENR_REJECT_LETTER instead of USP_REJD_LETTER

Use DPW standard abbreviations for words 7 or more characters.

Calls to Database Objects

- All calls to database objects must be prefixed with the schema name. The proper format for database object calls is:
- [schema_name].{table_name] Example: PACSES.t_member

Collections, Record Types and Variables

- All collection types must be in lower case, with a "typ_" prefix.
- The variable of a given type must be in lower case with a table or rec suffix indicating whether it is a table type or a record type.
- Example below:

```
TYPE typ_temp_table IS TABLE OF VARCHAR2(10) INDEX BY BINARY_INTEGER;
    temp_table      typ_temp_table;
```

Constants

- The constant name must begin with a C_.
- All constant names in uppercase.

Cursors

- The cursor name must begin with a cur_.
- The name must be in lowercase.
- The cursor input parameters follow the parameter naming convention.

Functions

- Names must start with FN_ and should be all uppercase.
- The keywords must be separated by an underscore.

Packages

- Names must start with PKG_ and should be all uppercase.
- The keywords must be separated by an underscore.

Parameters

- All parameters names in lowercase.
- Parameters names must begin with p_.
- The keywords must be separated by an underscore.
- If a parameter corresponds to a column in a table, name as follows: p_[column name in table]. Example: p_idn_member.
- Declare the parameter data type as corresponding column data type. Example: p_idn_member IN PACSES.t_member.idn_member%TYPE
- Always specify if the type of parameter as IN/ OUT /IN OUT.
- Specify the parameters for a program unit in the following order:

  1. IN 2. IN OUT 3. OUT

- Non-standard parameter name examples:
  P_STRSTAFFDOCTYPE
  P_STRHOWMANYDOCS
- Standard parameter name examples:
  p_str_staff_doc_type
  p_str_how_many_docs

Procedures

- Names must start with USP_ and should be all uppercase.
- The keywords must be separated by an underscore.
- For stored procedures called by .NET Data Access Logic Components, choose stored procedure names that emphasize the Data Access Logic Component to which they pertain. For example, a .NET Data Access Logic Component called Customer could use USP_CUSTOMER_INSERT and USP_CUSTOMER_UPDATE in a package called PKG_CUSTOMER.

Triggers

- Concatenation of a trigger identity string of “TR”, a one-character trigger event code, a two-character trigger operation code, a two-character trigger type code, and a descriptive string of table specific information related to the trigger.
- The concatenation is undertaken by an underscore between the five portions of the name.
- Trigger Event Code
  - “B” for BEFORE trigger event
  - “A” for AFTER trigger event
- Trigger Operation Code
  - “IN” for INSERT operation

- "UP" for UPDATE operation
- "DE" for DELETE operation
- "NS" for INSTEAD operation (redirect statement to effect other objects INSTEAD of the object listed
- "LO" for database level operations that occur upon logon
- "LF" for database level operations that occur upon logoff
- "ST" for database level operations that occur upon database startup
- "SD" for database level operations that occur upon database shutdown
- "ER" for operations that occur upon detection of an error condition
- "SC" for schema level operations (typically DDL execution)

- Trigger Type Code
  - "RW" for ROW LEVEL trigger type (triggers that are executed for every row effected during the transition
  - "ST" for STATEMENT LEVEL trigger types (triggers that execute only one for a particular transaction no matter how many rows are effected)
- Example below:
  A trigger with BEFORE event, for INSERT operation, of ROW LEVEL type, and acting on a table of APPLN
  TR_B_IN_RW_APPLN

Variables (Except for Collections and Records)

- All variable names in lowercase.
- Variable names must start with v_.
- The keywords must be separated by an underscore.
- If a variable corresponds to a column in a table, name as follows: v_[column name in table]. Example: v_idn_member.
- Declare the variable data type as corresponding column data type.
- Non-standard variable name examples:
  v_SelWhereQuery
  HISTSEQ
  NbrSeqAgrmt
  DteCompltAgrmt
- Standard variable name examples:
  v_sel_where_query
  v_hist_seq
  v_nbr_seq_agrmt
  v_dte_complt_agrmt

Formatting and Commenting Conventions

Good formatting should meet the needs of both the code developer and maintainer. The reason for formatting code is to reveal and reinforce the logical structure of the program. That said, it is more important to use formatting to demonstrate the intent of the program, not to make the code pleasing to the eye. Programs using a good standard format are much easier to read and maintain than those not following a standard format.

- Comment as you code. Commenting while the code is being written allows for better continuity and fewer bugs. It provides a logic check on the developer. Comments should be written for each block of code.
- Explain why. Do not repeat what is occurring in the code. Explain the business logic of the query, the basic intention of what was done. Answer the question, if I were taking over maintenance, would I have enough information to understand the business reason for (why) this stored procedure was written and what it is to accomplish.
- Make comments easily maintainable. Comments should be basically formatted to minimize future maintenance. Add the comment(s) BEFORE the SQL statement. The comment must be in its own block and not intermixed with the SQL statement.
- Maintain indentation. Comments reinforce indentation standards and therefore the logical structure of the program. Always indent comments at the same level of the code which they describe.

Comments in Stored Procedures: Summary, Maintenance Log, In-stream Documentation

The summary is placed at the top of a stored procedure and includes:
Name of stored procedure, any input/output parameters, Calls to other stored procedures,
Description that contains information on the logic, various modules of the program, and functionality of the stored procedure.

The summary also contains the maintenance log that includes:
Date of creation / amendment, the person who did the creation / amendment, version number, PCR number, and an explanation that includes a reason for the initial release or PCR. The PCR number must be referenced with a main explanation of what the PCR is intended to do.

```
/*****************************************************************************
//
// Name: xxx_xxxx_xxxx
//
// Parameter(s) input:
//     Department Number
// Parameter(s) output:
//     Number of employees
//
// Calls: (other stored proc)
//
// Description: Returns the number of employees assigned to a given
//              dept. Errors on invalid department. Departments with
//              zero employees is NOT an error.
//
//  Maintenance log:
//  Date      Created /      Change
```

```
//          amended by Control Program history
//
// 10/12/1999 Jane Doe       1.0.0   Initial release. Provide an overview of what stored procedure    //
does.
// 12/12/1999 John          1.0.1    PCR #:  Provide explanation of what the PCR is intended to do.  //
what was done to resolve the issues addressed in the PCR.
*****************************************************************/
```

In-stream Documentation

Add comments before the SQL statement.
-- Double dash is for a comment regarding the code following the comment

The comment must be in its own block and not intermixed with the SQL statement.
CREATE PACKAGE trans_data AS -- bodiless package

/* slash asterisk …… asterisk slash */
/*
Explain the business logic of the query, the basic intention of what was done, what is to be accomplished.
*/

Variable Declarations

- The following example shows the variable declaration standard:

```
v_str_case_exists           VARCHAR2(1) := 'N';
v_nbr_seq_cmt_case          NUMBER(7) := TO_NUMBER(p_nbr_seq_cmt_case);
v_newdate                   DATE;
```

Parameters

- Parameters must be stacked unless less than three.
- The following example shows the parameter declaration standard:

```
PROCEDURE USP_GET_CLNT_CMMNT_LS (
   p_idn_case     IN   VARCHAR2,
   p_num_days     IN   VARCHAR2,
   p_start_date    IN   VARCHAR2,
   p_end_date     IN   VARCHAR2,
   ref_cursor     OUT   ref_cursor_type) IS
```

Parentheses

- Leave a space before a parameter or column list, my_proc (i_idn_case, o_num_days).
- Leave a space between consecutive parentheses, like (((instead of (((.
- In stacked lists, the opening parenthesis is above the first item's line.
- In stacked lists, the closing parenthesis is on the last item's line.
- In the example below, see the opening and closing parentheses:

```
PROCEDURE USP_GET_CLNT_LS (
   p_idn_case          IN   VARCHAR2,
   p_num_days          IN   VARCHAR2,
   p_start_date        IN   VARCHAR2,
   p_end_date          IN   VARCHAR2,
   ref_cursor          OUT   ref_cursor_type) IS
```

Commas

- In stacked lists, use trailing commas. See above example.

Indenting

- Indent to 1 tab character (or 4 characters).
- Position THEN on the same line as IF.
- Position LOOP on the next line.
- Position IS (or AS) on the same line.

Linefeeds

In PL/SQL code

- At least one newline:
  - Around a DML statement.
  - Around a type declaration.
  - Around a program unit declaration.
  - Around a cursor declaration.
  - Around a program unit body.
  - Around an IF statement.
  - Around a LOOP statement.
  - Around a PL/SQL block.
  - Before a label.
  - Insert a newline between a specified construct and its preceding and subsequent constructs.

In SQL code:

- At least one newline:
  - Around program units.
  - Around anonymous blocks.
  - Around views.
  - Around triggers.
  - Around object types.
  - Around all other commands such as insert, update, delete, select.

Margins

- Left margin at 1 and right margin at 78.
- It is allowed to overshoot right margin if code cannot be accommodated in 78 characters.

Tabs

- Use tab instead of space.
- Tab should be set to 4.

Spacing

- Use white space inside a statement.
- Always include a space between an identifier and a separator.
- Use spaces to make module calls and their parameter lists more understandable.

Case

PL/SQL code is comprised of different components: variables, report fields, procedures, functions, etc. To improve readability of these components, it is beneficial to reflect a difference in the way code is displayed. The upper and lowercase strategy is recommended to distinguish between components.

Built-in Functions

- All built-in functions are in uppercase.
- Example below:
  like v_id := TO_CHAR(v_num);

Built-in Packages

- All built-in package names in uppercase.
- Example below:
  like DBMS_OUTPUT.PUT_LINE(v_out);

Keywords

- All SQL and PL/SQL keywords in uppercase.

Lists and Operators

AND-OR

- Always stacked.
- Operators left aligned, all conditions aligned when stacked.
- Example below:

```
SELECT    dte_change_last,
          idn_user_change_last,
          nbr_seq_log_asst,
          cnt_est_hour,
          tme_est_minute,
          txt_cmt,
          dte_rq
FROM      ccmis.t_provr_loc_asst_log
WHERE     dte_rq BETWEEN strbegindate AND strenddate
AND       idn_entity_legal_provr = p_strlegentity
AND       idn_loc_provr = p_lng_provrid
ORDER BY dte_rq ASC;
```

Plus-Minus-Multiply-Divide-Concatenate

- Always stacked.
- Operators left aligned, all expressions aligned when stacked.
- Example below:

```
-- Calculate Carelevel Age in Months
IF v_nbr_age IS NULL THEN
        -- Number of Months from December for V_DTE_AGE
        v_dte_age_month := TO_CHAR (v_dte_age, 'MM');
   v_dte_age_month :=   12
                     - v_dte_age_month;
        -- Number of Months from January for P_DTE_PROCESS
   v_dte_process_month := TO_CHAR (v_dte_process, 'MM');
        -- Difference in Years
   v_nbr_year :=   v_dte_process_year
                 - v_dte_age_year;
   v_nbr_month :=   12
                  * v_nbr_year;
        -- Sum Months Together
   v_nbr_age :=   v_nbr_month
                + v_dte_process_month
                + v_dte_age_month;
END IF;

SELECT UPPER (RPAD (
          SUBSTR (
             RTRIM (DECODE (
                       nam_last,
                       NULL, ' ',
                          nam_last
                       || ',')
                    || DECODE (
                       nam_first,
                       NULL, ' ',
                          nam_first
                       || ',')
                    || DECODE (
                       nam_mi,
                       NULL, ' ',
                       nam_mi),
                 ','),
              1,
              19),
           19,
           ' ') )
FROM   ccmis.t_provr_loc_cntc tplc
WHERE  tplc.idn_entity_legal_provr = p_leid
AND    tplc.idn_loc_provr = p_loc_id
AND    tplc.ind_record_curr = 'Y'
AND    tplc.nbr_seq_cntc =
       (SELECT MIN (tplco.nbr_seq_cntc)
        FROM   ccmis.t_provr_loc_cntc tplco
        WHERE  tplco.idn_entity_legal_provr = p_leid
        AND    tplco.idn_loc_provr = p_loc_id);
```

Specific Statements

Assignments

- Compact arrangements are used for assignments. The assignment operator is separated from the variable name by one space on the left hand side and by one space from the expression on the right hand side.
- Example below:

```
PROCEDURE USP_GET_PROVR_ASST_DTL (
   p_user_id              IN      VARCHAR2,
   p_lng_provr_id         IN      VARCHAR2,
   p_str_begin_date       IN      VARCHAR2,
   p_str_end_date         IN      VARCHAR2,
   p_str_leg_entity       IN      VARCHAR2,
   ref_cursor             OUT     ref_cursor_type) IS
   v_str_begin_date               DATE;
   v_str_end_date                 DATE;
BEGIN
   v_begin_date := TO_DATE (p_str_begin_date, 'mm/dd/yyyy');
   v_end_date := TO_DATE (p_str_end_date, 'mm/dd/yyyy');
   v_eff_date := TO_DATE (p_str_begin_date, 'mm/dd/yyyy');
   v_last_change_date := TO_DATE (p_str_end_date, 'mm/dd/yyyy');
.
.
.

END USP_GET_PROVR_ASST_DTL;
```

SELECT

- Keywords are left aligned.
- The column list, INTO variable list, table list, order by list, and group by list are stacked.
- Example below:

```
/* Formatted on 2003/03/11 13:15 (Formatter Plus v4.7.0) */
SELECT   SUM (cnt_cap_lic) cnt_cap_lic,
         SUM (cnt_enroll_desire) cnt_enroll_desire,
         SUM (cnt_enroll_sbsd_desire) cnt_enroll_sbsd_desire,
         MAX (idn_user_change_last) idn_user_change_last,
         MAX (dte_change_last) dte_change_last
FROM     (SELECT idn_entity_legal_provr,
                 idn_loc_provr,
                 cnt_enroll_desire,
                 cnt_enroll_sbsd_desire,
                 cnt_cap_lic,
                 idn_user_change_last,
                 NVL (TO_CHAR (dte_change_last, 'MM/DD/YYYY'), '01/01/2000')
                                                  dte_change_last
          FROM   ccmis.t_provr_loc_cap
          WHERE  idn_entity_legal_provr = p_idn_entity_legal_provr
          AND    idn_loc_provr = p_idn_loc_provr
          UNION
          SELECT idn_entity_legal_provr,
                 idn_loc_provr,
                 0,
                 0,
```

```
            cnt_cap_lic,
            'A',
            '01/01/2000'
      FROM   ccmis.t_provr_loc_lic
      WHERE  idn_entity_legal_provr = p_idn_entity_legal_provr
      AND    idn_loc_provr = p_idn_loc_provr
      AND    ind_record_curr = 'Y'
      AND    ind_record_delete_logcl = 'N'
      AND    TRUNC (SYSDATE) BETWEEN dte_issue AND dte_expire)
GROUP BY idn_entity_legal_provr,
      idn_loc_provr;
```

INSERT

- Keywords are left aligned.
- All parameter and variable lists are stacked.
- Example below:

```
INSERT INTO ccmis.t_provr_loc_early_cc_prog
        (
        idn_entity_legal_provr,
        idn_loc_provr,
        cde_prog_cc_early,
        dte_change_last,
        idn_user_change_last)
VALUES    (
        p_idn_entity_legal_provr,
        p_idn_loc_provr,
        newcdeprogccearly,
        SYSDATE,
        p_idn_user_change_last);
```

UPDATE

- Keywords are left aligned.
- All parameter and variable lists are stacked.

DELETE

- Keywords are left aligned.
- All parameter and variable lists are stacked.

SQL Statement Construction

Table Aliases

- To enhance the readability of your SQL statement (when more than one table is involved), you may want to construct alias names by taking the first letter of each word in a table name. This is a general guideline and can be deviated from when needed to enhance the meaning of the alias.
- Alias names should use the first letter of each word in the table name, using additional letters as necessary.

- Do not use t1, t2, etc.
- Examples below:
  T_CLIENT → C
  T_CLIENT_ADDRESS → CA

Placement of Join and Criteria in WHERE Clause

- Place your join first in the WHERE clause, then your criteria.

SQL Injection

Guard against SQL injection by using bind variables. Avoid using dynamic SQL, but when dynamic SQL is used, verify the input for accuracy.

Testing

Write test scenarios for testing each stored procedure before writing any lines of code. A clear understanding of what the code is to do is essential prior to writing the code.

Test, debug and evaluate the performance of each stored procedure with volumes of data that match production volumes. There is no substitute for volume testing.

General Standards

- Do not use SELECT *. Instead list the column names needed in the SELECT statement.
- Use VARCHAR2 instead of CHAR. The CHAR data type is padded with spaces for data items with varying widths and requires using rpad() to obtain accurate search results
- Do not try to do many things with one SQL statement. There is no harm in writing 5 SQL statements instead of trying to create a complicated query to get all data with one query having a 5 table join. Intermediate tables can be used for temporary data storage.
- Think. Is there a better way to store data so it can be retrieved more efficiently? Is each line, each function in my code absolutely necessary to get the data? Can a built-in functionality or a built-in feature be used to retrieve data more efficiently?

ORACLE 10g PL/SQL Features

Oracle 10g PL/SQL Features

- The binary_float and binary_double datatypes (the IEEE datatypes).
- The regexp_like, regexp_instr, regexp_substr and regexp_replace builtins to support regular expression manipulation with standard POSIX syntax.
- Multiset operations on nested table instances supporting operations like equals, union, intersect, except, member, and so on.
- The user-defined quote character.
- Indices of and values of syntax for forall.
- The distinction between binary_integer and pls_integer vanishes.

Oracle 10g PL/SQL Supplied Packages

- Utl_Mail: This new package makes it possible for a PL/SQL programmer to send programmatically composed emails.
- Utl_Compress. This new package delivers the familiar functionality of the zip and unzip utilities in a PL/SQL environment. It lets you compress and uncompress a raw or blob bytestream and guarantees return of original bytestream after a round trip.

- Dbms_Warning. This allows the PL/SQL programmer fine grained control over which categories of warning and which individual warnings to disable, to enable, or to treat as errors.

Using Oracle Database 10g PL/SQL New Features Tutorial

http://www.oracle.com/technology/obe/obe10gdb/develop/plsql/plsql.htm#o3 In this tutorial, you learn about the new PL/SQL features that are introduced in Oracle Database 10g.

DBMS_OUTPUT

SQL*Plus will now display your DBMS_OUTPUT after the result of a SELECT statement from procedures, triggers, and functions. This is great news because now DBMS_OUTPUT displays right after the execution of a SELECT statement and does not require you to take additional action for this display.

Bulk Binding Enhancements

Oracle Database 10g extends the enhancements that were introduced in Oracle9i in the area of bulk binding. The SAVE EXCEPTIONS syntax was introduced in Oracle9i to capture exceptions while bulk inserting (deleting or updating) rows. Although this feature enables the data manipulation language (DML) to continue (saving any exceptions in the SQL%BULK_EXCEPTIONS collection), the performance of the operation is greatly affected. In a scenario in which the collection being processed is sparse due to the application logic involved, this would be an unnecessary overhead. In Oracle Database 10g, you can overcome these issues with the new INDICES OF and VALUES OF features.

The INDICES OF keyword can be used in a scenario in which a collection of records (which is dense) is validated programmatically and invalid records (those not meeting the specified criterion) are removed from the collection. This result is a sparse collection of valid elements that must then be bulk inserted into a table. By using the INDICES OF keyword, the exceptions for the missing records are not generated.

The VALUES OF keyword can be used in a scenario in which a collection of records (sparse or dense) must be copied to one or more collection variables, based on some condition whereby certain records may or may not be copied, and then inserted into a table. This can be efficiently done by using the "VALUES OF" syntax and using a pointer array whose elements are pointers to the selected records within the original collection. This reduces the need to create multiple copies of data. Exception handling is done per pointer record, which means that if two or more pointer records are pointing to the same "original" data, any exceptions that are reported indicate the iteration number (pointer element number).

Debugging PL/SQL with JDeveloper

Starting with version 9.0.3, JDeveloper supports PL/SQL debugging. This is achieved with the Java Debugging Wire Protocol (JDWP). With Oracle9i, release 2 (and later versions), the debugger uses the industry standard, whereas for earlier releases (Oracle8i and Oracle9i, release 1), the DBMS_DEBUG package is used. The debugging UI gives you the ability to single step, step into, and step over PL/SQL objects as well as the ability to set PL/SQL expressions in the Watch and Inspector windows, set conditional breakpoints, and view collection data.

PL/SQL Compiler Enhancements

The PL/SQL compiler is substantially improved in Oracle Database 10g. The back-end code generator is reengineered. Also, the new parameter PLSQL_OPTIMIZE_LEVEL has been introduced to speed up the execution of PL/SQL code. This is in addition to native compilation (which has been available since Oracle9i). The PLSQL_OPTIMIZE_LEVEL parameter default is currently "2".

- Expression in a Loop: Don't write an expression inside a loop if it remains a constant over all iterations of the loop. Was good advice, but in 10g no need to care.
- If Test on a Constant: Declaring a variable as CONSTANT has no impact on performance. Was true, but in 10g using CONSTANT improves performance.
- Breaking big expressions into pieces: Avoid successive assignments to work variables. Collapse these into a single assignment. In 10g, go for clarity and correctness.
- Declare... End Blocks inside Loops: In 10g, go for clarity and correctness.
- Collection Indexing: Some hand optimization used to be beneficial, some disastrous. In 10g the new PL/SQL optimizing compiler manages all optimizations.
- Implementing callback: You have to use dynamic SQL to implement a callback. This will cost hugely in call overhead. Was true before 9iR1. In 10g polymorphism helps performance.
- SELECT to Fetch Exactly One Row: Use OPEN-FETCH-CLOSE, it's quicker than SELECT INTO is a myth. Worth weeding this out of the code, especially for native dynamic SQL case.
- Dynamic SQL versus static SQL: Dbms_sql is never the winner. Use NDS. NDS single-row was slower than static-row in 9iR2, but they are similar in 10gR1 because of cursor reuse. NDS and static SQL were similar for the nonbulk case in 9iR1, but static leaps ahead in 10gR1 because the PL/SQL optimizing compiler bulkifies it. NDS and static SQL are similar for the bulk case.
- Dynamic SQL versus static SQL: Neither static SQL nor Dynamic SQL avoid soft parse when using a REF CURSOR. New in 10gR1, native dynamic SQL avoids soft parse when not using a REF CURSOR. For single row select and for bulk syntax, native dynamic SQL is therefore about as fast as static SQL.

- Nonbulk select versus bulk select:: Oracle Database 10g's optimizing PL/SQL compiler will bulkify nonbulk SQL behind the scenes. Not true. Not there yet, but is the direction. So far, the optimizing compiler bulkifies only the static SQL cursor for loop. Don't throw this chance away by using the open, fetch loop, close form. Bulk syntax continues to matter for dynamic SQL cases.

APPENDEX I – GLOSSARY OF PL/SQL TERMS

Array - In computer programming languages, an array is a group of objects with the same attributes that can be addressed individually

Assignment Statement - An assignment statement sets the current value of a variable, field, parameter, or element. The statement consists of an assignment target followed by the assignment operator and an expression. When the statement is executed, the expression is evaluated and the resulting value is stored in the target

Blocks - The basic program unit in PL/SQL is the block. A PL/SQL block is defined by the keywords DECLARE, BEGIN, EXCEPTION, and END. These keywords partition the block into a declarative part, an executable part, and an exception-handling part. Only the executable part is required. You can nest a block within another block wherever you can place an executable statement.

Concurrency - simultaneous access of the same data by many users. Oracle manages concurrency by using various types of locks and a multiversion consistency model.

Cursors - To execute a multi-row query, Oracle opens an unnamed work area that stores processing information. An explicit cursor lets you name the work area, access the information, and process the rows individually.

DBA - A database administrator (DBA) directs or performs all activities related to maintaining a successful database environment. Responsibilities include designing, implementing, and maintaining the database system; establishing policies and procedures pertaining to the management, security, maintenance, and use of the **database management system**; and training employees in database management and use.

Database - A database is a collection of data that is organized so that its contents can easily be accessed, managed, and updated.

DBMS - A database management system (DBMS), sometimes just called a database manager, is a program that lets one or more computer users create and access data in a database. The DBMS manages user requests (and requests from other programs) so that users and other programs are free from having to understand where the data is physically located on storage media and, in a multi-user system, who else may also be accessing the data. In handling user requests, the DBMS ensures the integrity of the data (that is, making sure it continues to be accessible and is consistently organized as intended) and security (making sure only those with access privileges can access the data).

Loop – In computer programming, a loop is a sequence of instructions that is continually repeated until a certain condition is reached. Typically, a certain process is done, such as getting an item of data and changing it, and then some condition is checked such as whether a counter has reached a prescribed number. If it hasn't, the next instruction in the sequence is an instruction to return to the first instruction in the sequence and repeat the sequence. If the condition has been reached, the next instruction "falls

through" to the next sequential instruction or branches outside the loop. A loop is a fundamental programming idea that is commonly used in writing programs.

Packages - A package is a schema object that groups logically related PL/SQL types, items, and subprograms. Packages have two parts: a specification (spec for short) and a body.

Procedures - A procedure is a subprogram that can take parameters and be invoked. Generally, you use a procedure to perform an action. A procedure has two parts: the specification and the body. The specification (spec for short) begins with the keyword PROCEDURE and ends with the procedure name or a parameter list. The procedure body has three parts: an optional declarative part, an executable part, and an optional exception-handling part. The declarative part contains declarations of types, cursors, constants, variables, exceptions, and subprograms. These items are local and cease to exist when you exit the procedure. The executable part contains statements that assign values, control execution, and manipulate Oracle data. The exception-handling part contains handlers that deal with exceptions raised during execution.

Processing Types – OLTP, OLAP, Batch

OLTP – Online Transaction Processing executes during peak or prime time on live data. OLTP transactions provide on demand, immediate responses to the user.

OLAP – Online Analytical Processing is performed on servers containing only data warehouse or history data.

Batch – Batch Processing consists of longer running programs that execute on a batch or collection of user requests. Also, users are not needed for interaction. Batch processing is scheduled during off peak hours against live data.

Query - In general, a query (noun) is a question, often required to be expressed in a formal way. In computers, what a user of a search engine or database enters is sometimes called the query. To query (verb) means to submit a query (noun).

A database query can be either a select query or an action query. A select query is simply a data retrieval query. An action query can ask for additional operations on the data, such as insertion, updating, or deletion.

Relational database - A relational database is a collection of data items organized as a set of formally-described tables from which data can be accessed or reassembled in many different ways without having to reorganize the database tables.

Stored procedure - In a database management system (DBMS), a stored procedure is a set of Structured Query Language (SQL) statements with an assigned name that's stored in the database in compiled form so that it can be shared by a number of programs. The use of stored procedures can be helpful in controlling access to data (end-users may enter or change data but do not write procedures), preserving data integrity (information is entered in a consistent manner), and improving productivity (statements in a stored procedure only need to be written one time).

Trigger - In a database, a trigger is a set of Structured Query Language (SQL) statements that automatically "fires off" an action when a specific operation, such as changing data in a table, occurs. A trigger consists of an event (an INSERT, DELETE, or UPDATE statement issued against an associated table) and an action (the related procedure). Triggers are used to preserve data integrity by checking on or changing data in a consistent manner.

SQL Tuning - SQL tuning is the process of ensuring that the SQL statements that an application will issue will run in the fastest possible time. Just like there may be ten different ways to drive from work to your house, there may be ten different ways to execute a query. Normally, you are in a hurry to get home so you take the fastest way. Once in a while, you may find a new way that is even faster. You did not consider this route before because it was not very intuitive. SQL tuning is similar. Tuning SQL statements is finding the fastest route to answer your question, even if that route is not very intuitive

Exemptions from this Standard:

ITASCA and Oracle APEX databases and Commercial Off-The-Shelf (COTS) databases.

Refresh Schedule:

All standards and referenced documentation identified in this standard will be subject to review and possible revision annually or upon request by the DPW Information Technology Standards Team.

Standard Revision Log:

Change Date	Version	Change Description	Author and Organization
08/14/2003	1.0	Created Document	Susan Pracht
08/20/2007	2.0	Reviewed and revised document	Patty Gillingham
10/20/2009	3.0	Added ITB reference moved Best Practices and Helpful Hints to Oracle PL SQL Guidelines	Patty Gillingham