SIGPRE – Sistema de Gestión Presupuestaria

Guía de Programación Oracle

Histórico de Revisiones

Fecha	Versión	Descripción	Autor
10/2/2007	1.0	Borrador	Roberto López
11/14/2007	1.0	Convenciones Objetos de BDD	Roberto López
11/15/2007	1.0	Convenciones Forms	Roberto López
11/16/2007	1.0	Convenciones Reports	Roberto López
11/26/2007	1.1	Revisión del personal técnico del centro de cómputo.	Centro de Cómputo
11/26/2007	1.1	Se agrega convenciones para uso de Journal tables y TABLE API de Oracle Designer	Roberto López

Tabla de Contenidos

Tabla de Figuras

1. Introducción

Con el fin de estandarizar el desarrollo sobre la plataforma Oracle que se realice en el Centro de Cómputo de EMELNORTE S.A., la presente guía establece recomendaciones para la programación con PL/SQL

1.1.Propósito

El propósito de ésta guía es establecer un estándar o convención de nombrado para el desarrollo de aplicaciones bajo la plataforma Oracle, específicamente para Objetos de bases de datos y PL/SQL

1.2.Alcance

Aplicable al desarrollo de aplicaciones sobre plataforma Oracle:

- Programación PL/SQL
  - Objetos de Base de Datos
    - Tablas, vistas, índices
    - Triggers, Paquetes, Funciones y Procedimientos

2. Organización de Código y Estilos

1.3.Estándares para formato de código

1.3.1. Capitalización

Todas las palabras clave SQL deben estar en mayúsculas. Cualquier palabra clave Oracle o “built-ins” deberán también ser escritas en mayúsculas. No use capitalización mezclada.

Este método facilita diferenciar entre una llamada a procedimientos “built-in” de una llamada a procedimientos programados.

1.3.2. Identación

Use tres espacios (en lugar de TAB) en los siguientes casos:

- Para código entre las sentencias BEGIN y END
- Sentencias LOOP y END LOOP
- Para el código entre las sentencias IF, ELSE y ELSE IF

Se recomienda el uso de tres espacios en lugar de tabuladores, por las siguientes razones:

- Los Tabuladores no son portables entre editores.
- Los Tabuladores no almacenan el código de manera apropiada en la Base de datos

El código fuente de la Base de datos puede ser visto usando el diccionario de datos, a través de las vistas USER_SOURCE, ALL_SOURCE, o DBA_SOURCE. Estas vistas no despliegan apropiadamente el código que ha sido formateado con tabuladores. Esto dificulta la visualización de código almacenado sin necesidad de extraerlo a un archivo.

Tabuladores usualmente son muy largos, algunos editores como Notepad y otros usan 8 espacios para desplegar un carácter de tabulación.

Se recomienda el uso del editor del programa TOAD para la programación sobre Oracle. "TOAD for Oracle" es una herramienta que permite especificar opciones de formato que se aplicarán a las unidades de programa que sean desarrolladas. Estos parámetros de formateo de código deben configurarse de manera que cumplan con las recomendaciones del presente documento.

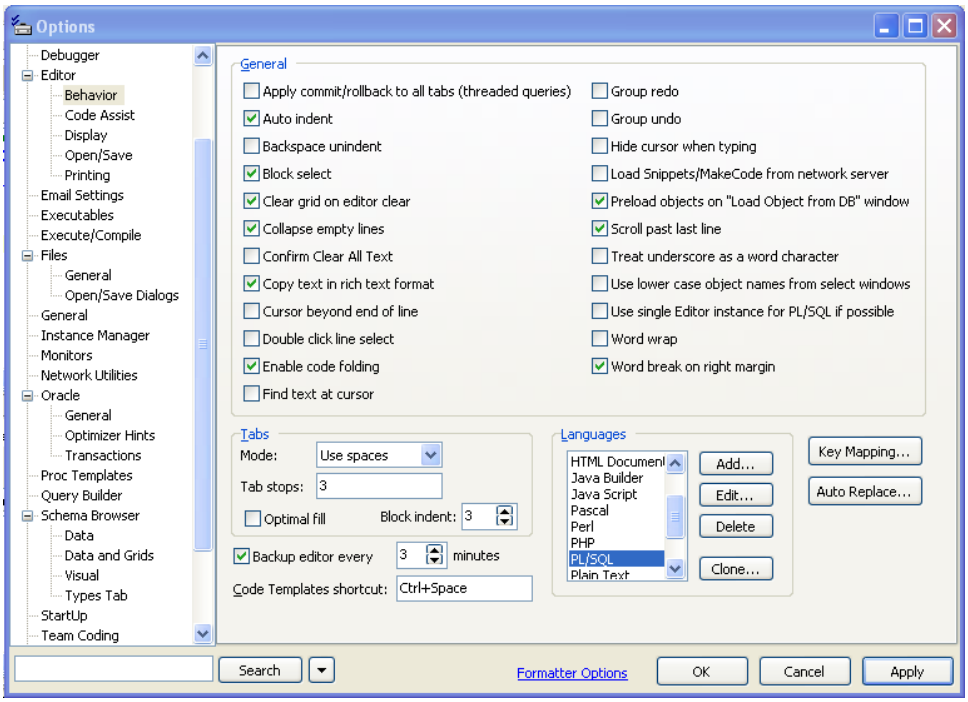

Figura 1 Opciones del Editor de "TOAD for Oracle"

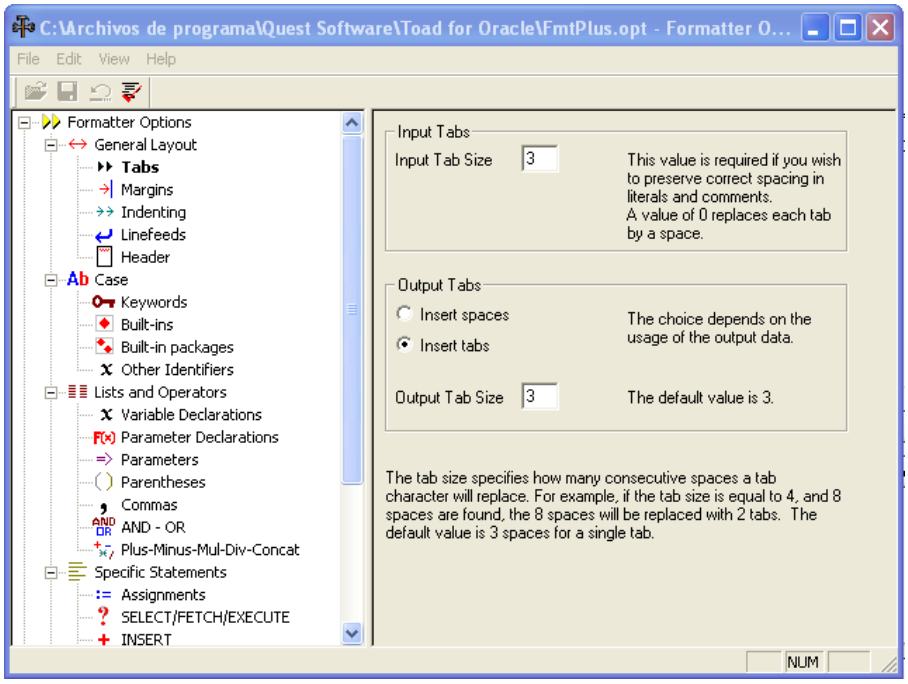

Figura 2 Opciones de Formato de código en "TOAD for Oracle"

Otra opción es el uso de editores gratuitos que permitan parametrizar el numero de caracteres para cuando se usa la un tabulador. Ejemplo:

PFE (Programmer's File Editor). http://www.lancs.ac.uk/people/cpaap/pfe/

1.3.3. Alineación

Considere las opciones de alineación en el formateo de código que realiza el editor de TOAD.

Además, considere las siguientes recomendaciones:

Sentencias DECLARE, BEGIN, EXCEPTION, and END deberán alinearse a la izquierda.

Las palabras claves de una sentencia SQL deben estar alineadas a la izquierda, cláusulas GROUP BY, ORDER BY, DELETE FROM, INSERT INTO, etc.

Cuando tenga más de una sentencia condicional o sentencias de asignación, alinee los elementos para cada condición Por ejemplo: El ejemplo presentado más adelante contiene una sección que muestra la alineación de este tipo de sentencias.

En declaraciones de procedimientos y funciones, alinee los nombres de parámetros cada uno a la izquierda así como también los tipos de datos de cada parámetro. Observe el ejemplo siguiente.

1.1.1.1. Ejemplos de sentencias SQL:

```
SELECT e.numero_etiqueta, e.peso, e.numero_piezas
  FROM etiquetas e, localizaciones l, estados_localizaciones el
 WHERE     e.codigo_localizacion = l.codigo_localizacion
       AND l.numero_sitio = i_numero_sitio
       AND l.estado_localizacion = el.estado_localizacion
       AND l.bandera_venta = 'S'
       AND l.tipo_localizacion = 'F'
       AND e.numero_parte = i_numero_parte
       AND e.contrato = i_contrato
       AND e.numero_piezas <> i_numero_piezas
       AND e.numero_etiqueta NOT IN (SELECT numero_etiqueta
                                       FROM ordenes
                                      WHERE estado = 'ACTIVO');
```

```
UPDATE ordenes
   SET estado = 'ACTIVO', bandera_envio = 'Y'
 WHERE     numero_etiqueta = l_numero_etiqueta
       AND numero_orden = i_numero_orden
       AND codigo_orden = i_codigo_orden
       AND numero_linea = i_numero_linea;
```

```
INSERT INTO ordenes (numero_orden, codigo_orden, numero_linea)
    VALUES ('123456', 'O', 1);
```

```
DELETE FROM ordenes
      WHERE     numero_orden = i_numero_orde
            AND codigo_orden = i_codigo_orden
            AND numero_linea = i_numero_linea;
```

1.1.1.2. Ejemplos de alineación en programas

```
/* Formatted on 10/13/2010 6:46:59 PM (QP5 v5.163.1008.3004) */
CREATE OR REPLACE PROCEDURE pro_crear_envio (
   i_numero_orden   IN     ordenes.numero_orden%TYPE,
   io_parametro     IN OUT NUMBER)
IS
   CURSOR l_cur_ordenes
   IS
      SELECT numero_orden, codigo_orden, numero_linea
        FROM detalle_ordenes
       WHERE numero_orden = i_numero_orden;

   l_reg_ordenes   l_cur_ordenes%ROWTYPE;
BEGIN
   OPEN l_cur_ordenes;

   FETCH l_cur_ordenes INTO l_reg_ordenes;
```

```
  CLOSE l_cur_ordenes;

  FOR l_contador IN 1 .. 10
  LOOP
    INSERT INTO envios_ordenes (numero_orden,
                                código_orden,
                                numero_linea,
                                numero_envio)
         VALUES (l_reg_ordenes.numero_orden,
                 l_reg_ordenes.codigo_orden,
                 l_reg_ordenes.numero_linea,
                 l_contador);
  END LOOP;
EXCEPTION
  WHEN OTHERS
  THEN
    raise_an_error;
END pro_crear_envio;
```

Ejemplo de una sentencia con mala alineación:

```
l_mensaje := 'La solicitud número ' || i_numero_orden || ' solicitada por ' || i_nombres_empleado || '
ha sido enviada con fecha ' || TO_CHAR(l_date_created, 'MM/DD/YYYY');
```

Un mejor alineamiento sería:

```
l_mensaje :=
        'La solicitud número '
     || i_numero_orden
     || ' '
     || 'solicitada por '
     || i_nombres_empleado
     || ' '
     || 'ha sido enviada con fecha '
     || TO_CHAR (l_fecha_solicitud, 'MM/DD/YYYY');
```

El código que no se rige a las convenciones de alineación puede resultar complicado en su lectura por parte de un programador:

```
SET_ITEM_PROPERTY('block1.item1', ENABLE, PROPERTY_TRUE);
SET_ITEM_PROPERTY('block100.item100', ENABLE, PROPERTY_TRUE);
SET_ITEM_PROPERTY('block10.item2', ENABLE, PROPERTY_FALSE);
SET_ITEM_PROPERTY('block10000.item3', ENABLE, PROPERTY_TRUE);
SET_ITEM_PROPERTY('block1.item2', ENABLE, PROPERTY_TRUE);
SET_ITEM_PROPERTY('block100.item101', ENABLE, PROPERTY_TRUE);
SET_ITEM_PROPERTY('block10.item3', ENABLE, PROPERTY_TRUE);
SET_ITEM_PROPERTY('block10000.item4', ENABLE, PROPERTY_TRUE);
```

Una adecuada alineación facilita la lectura del código:

```
SET_ITEM_PROPERTY('block1.item1',        ENABLE,  PROPERTY_TRUE);
SET_ITEM_PROPERTY('block100.item100',  ENABLE,  PROPERTY_TRUE);
SET_ITEM_PROPERTY('block10.item2',       ENABLE, PROPERTY_FALSE);
SET_ITEM_PROPERTY('block10000.item3',  ENABLE,  PROPERTY_TRUE);
SET_ITEM_PROPERTY('block1.item2',        ENABLE,  PROPERTY_TRUE);
```

```
SET_ITEM_PROPERTY('block100.item101',  ENABLE,  PROPERTY_TRUE);
SET_ITEM_PROPERTY('block10.item3',     ENABLE,  PROPERTY_TRUE);
SET_ITEM_PROPERTY('block10000.item4',  ENABLE,  PROPERTY_TRUE);
```

Al definir o invocar procedimientos con múltiples parámetros, ponga cada parámetro en una línea diferente:

```
PROCEDURE obtener_orden (numero_orden IN detalle_ordenes.numero_orden%TYPE,
                         codigo_orden  IN detalle_ordenes.codigo_orden%TYPE,
                         numero_linea  IN detalle_ordenes.numero_linea%TYPE);
```

```
obtener_orden ( l_numero_orden
              ,l_codigo_orden
              ,l_numero_linea );
```

Las comas deben ser ubicadas al inicio de la línea en el caso de separar argumentos, parámetros u otras listas encerradas en paréntesis.

Las comas deben ser ubicadas al final de la línea cuando son usadas para separar elementos no contenidos entre paréntesis, como en una lista de columnas en una sentencia SELECT.

3. Comentarios

Incluya un comentario como encabezado en cada procedimiento y función. El encabezado debe ser actualizado en cada modificación realizada al código. El encabezado debe ser un resumen de la unidad de programa, por ejemplo:

```
/*
-------------------------------------------------------
Autor: R López
Propósito: Crear un Nuevo registro en la tabla empleados
Estructura lógica: Documente el flujo básico del programa en alto nivel.
Historial de Modificaciones:
Fecha       Nombre       Resumen de Revisión
----------  ---------- -----------------------------------
11/09/2007  R. López    Creación
11/10/2007  M. Rea      Añadido código de seguridades
-------------------------------------------------------
*/
```

Una recomendación para el caso de los cometarios es usar los “wizard” de creación de programas de TOAD, el cual agrega un encabezado para en cada creación.

Use comentarios para explicar la lógica compleja en sus programas. En general, si se siguen las convenciones de manera adecuada y se escribe código de manera modular, la programación será auto descriptible, es decir, no será necesario agregar demasiados comentarios.

Una recomendación es no repetir en los comentarios lo que el código por si solo puede describir, por ejemplo:

```
--si el código de estado es nulo entonces lanzar una excepción
IF codigo_estado IS NULL THEN
   RAISE FORM_TRIGGER_FAILURE;
END IF;
```

El ejemplo anterior es un comentario inútil, pues el código escrito describe la acción que se está ejecutando.

El siguiente es un ejemplo de cómo un comentario ayuda a describir un cálculo complejo:

```
/* 01/10/2007. Cálculo añadido para corregir errores de redondeo causados por el hecho de que se
almacenan valores redondeados a 3 decimales. Básicamente el código incrementa en número de
dígitos */
SELECT ( ( (ROUND (96 * (length_no - FLOOR (length_no))) / 96)
         + FLOOR (length_no))
       * cs.lbs_ft
       * i_qty)
         exact_length
  FROM cs_sizes cs, part_description pd
 WHERE     pd.part_no = i_part_no
       AND pd.contract = i_contract
       AND pd.size_code = cs.size_code;
```

4. Nombrado

Use nombres significativos. Evite el uso de nombres de variables pequeños o arbitrarios como "x" o "var". Éstos nombres no transmiten el contenido almacenado en la variable. Los nombres de los objetos dentro del código deben proporcionar una “auto-documentación” del mismo. En otras palabras si nombramos los ítems de manera significativa implícitamente indicamos lo que están realizando dentro del código.

Si se está declarando una variable o un parámetro que haga referencia a un valor guardado en una columna de la base de datos, use el nombre de la columna para declarar la variable. Por ejemplo, v_nombre_empleado

Nunca use "a", "b", "c", etc. Como alias de tablas en sus sentencias SQL. En una consulta que involucre algunas tablas se dificultará su lectura, comprensión y depuración. Ejemplo:

```
SELECT a.fecha_orden
  FROM ordenes a, detalle_ordenes b, precios c
 WHERE a.numero_orden = b.numero_orden AND b.linea_orden = c.linea_orden;
```

Considere usar nombres significativos

```
SELECT ord.fecha_orden
  FROM ordenes ord, detalle_ordenes det, precios prc
 WHERE ord.numero_orden = det.numero_orden
       AND det.linea_orden = prc.linea_orden;
```

Una tercera opción es usar como alias las letras iniciales de los nombres de tablas

```
SELECT o.order_date
  FROM ordenes o, detalle_ordenes do, precios p
 WHERE o.numero_orden = do.numero_orden
    AND do.linea_orden = p.linea_orden;
```

El segundo y tercer ejemplos son aceptables, no así el primero. La tercera opción puede requerir pequeños ajustes cuando las iniciales de los nombres de las tablas involucradas en la sentencia SQL coincidan, pero funciona.

Use guiones bajos para separar palabras o identificadores. Por ejemplo: numero_orden_compra

Cuando se declare tipos de datos de variables o parámetros que correspondan a columnas de tablas de la base de datos, utilice el siguiente esquema:

```
nombre_tabla.nombre_columna%TYPE;
```

5. Declaración

1.4.Variables

El ámbito de las variables está determinado por un prefijo, así:

Variables locales	vnombre_variable	Incluye variables internas en funciones y procedimientos
Variables Globales	gnombre_variable	Variables a nivel de paquete
Constantes	cnombre_constante gc nombre_constante	
Cursores	cur_nombre_cursor	Note que los cursores pueden ser identificados como locales o globales tal como un nombre de variable
Registros (records)	rec_nombre_registro	Aplica a registros definidos por el programador tales como registros en base a cursores o tablas.
Tipos de cursores	ct_nombre_tipo_cursor	TYPE tcur_registro_mes IS REF CURSOR
Tipos de Registros (record type)	rt_nombre_registro	TYPE rt_registro_mes IS RECORD (
Tipos de Tabla PL/SQL	tt_nombre_tabla	TYPE tt_registro_mes IS TABLE OF rt_registro_mes INDEX BY BINARY_INTEGER;
Tablas PL/SQL	tnombre_tabla	
Tipos Varray	tv_nombre_arreglo	
Variables Arreglos	vnombre_variable	Para hacer referencia a un arreglo simpre necesitaremos indicar su índice v_numero(1)

1.5.Parámetros

1.5.1. Parámetros en funciones y procedimientos

De entrada (IN)	i_nombre_parametro	
De salida (OUT)	o_nombre_parametro	
De entrada y salida (in out)	io_nombre_parametro	

1.5.2. Parámetros en cursores

```
p_parametro_cursor
```

6. Expresiones y Sentencias

1.6.Procedimientos y Funciones

Al nombrar procedimientos, use una combinación de verbos y sustantivos que describan lo que el procedimiento realiza. Por ejemplo

```
pro_generar_rol
```

AL nombrar funciones, piense en como la función será invocada desde el código y nómbrela de acuerdo al contexto, por ejemplo, si el numero de cédula es no válido la función fun_cedula_valida retorna un valor BOOLEAN

```
fun_cedula_valida
```

```
IF fun_cedula_valida (codigo_empleado) THEN
   "acción";
END IF;
```

El uso de una función llamada fun_cedula_valida hace al código escrito más entendible, situación contraria si se usa por ejemplo:

```
IF fun_validar_cedula (codigo_empleado) THEN
      "acción";
END IF;
```

1.7.Bucles

Los bucles deben tener un solo punto de entrada y un solo punto de salida. Intente evitar el uso de múltiples puntos de salida. Esto provoca que el código se vuelva muy confuso.

Use etiquetas de bucles cuando trabaje con bucles anidados y cuando el código entre el inicio y fin del bucle sea muy extenso y provoque saltos de página. Ejemplos:

1.7.1. Bucles anidados sin etiquetas:

```
LOOP
   algunas sentencias;
   EXIT WHEN condicion_finalizacion_1;
   LOOP
```

```
      algunas sentencias que ocasionan saltos de pagina;
      EXIT WHEN condicion_finalizacion_2;
   END LOOP;
END LOOP;
```

1.7.2. Usando etiquetas de bucle:

```
<<loop_clientes>>
LOOP
   algunas sentencias que ocasionan saltos de página
   EXIT loop_clientes WHEN condicion_finalizacon_1;
   <<loop_ordenes>>
   LOOP
      algunas sentencias que ocasionan saltos de página
      EXIT loop_ordenes WHEN condicion_finalizacon_2;
   END LOOP loop_ordenes;
END LOOP loop_clientes;
```

1.7.3. For Loops

Los bucles FOR LOOP tienen la siguiente sintaxis:

```
FOR l_iteraciones IN 1..l_max_iteraciones LOOP
...

END LOOP;
```

Un bucle for loop debe ser usado cuando se requiere ejecutar un número específico de iteraciones

No obligue a salir del bucle hasta que el número de iteraciones se hayan ejecutado. No cambie el valor del índice del bucle para obligar una salida temprana del bucle.

1.7.4. Bucles While

Sintaxis:

```
WHILE l_condicion_booleana LOOP
END LOOP;
```

Los bucles while deben ser usados cuando se requiere ejecutar hasta que ocurra una condición específica (definida en la cláusula WHILE). No fuerce la salida del bucle hasta que la condición se haya cumplido.

Los bucles while son una mayor alternativa a los bucles for loop si se requiere una ejecución iterativa hasta que ocurra una condición. Simplemente se puede especificar una condición booleana a TRUE cuando necesite salir del bucle. No use bucles WHILE en lugar de bucles FOR. Ejemplo:

```
l_valor := 1;
WHILE l_valor <= 10 LOOP
...
END LOOP;
```

Si el bucle tiene una estructura como el anterior, use un bucle FOR en lugar de éste.

1.1.1.3. Bucles Simples

Sentencias como EXIT y EXIT WHEN son muy usadas en bucles con condiciones de salida complicadas. Específicamente, se debe usar bucles simples cuando requerimos una salida del bucle en un lugar diferente al inicio del mismo.

1.7.5. Cursores

Evite el uso de cursores implícitos. Use cursores explícitos

Un cursor explícito es aquel que se ha definido en el bloque de declaraciones. Un cursor implícito es aquel usado por Oracle para realizar un fetch cuando se ha escrito sentencias como SELECT … INTO .

Los cursores explícitos son más eficientes debido a que el programador controla manualmente el número de extracciones (fetches).

Ejemplo de cursor implícito:

```
PROCEDURE pro_obtener_nombre_cliente (
  i_codigo_cliente      IN     clientes.codigo_cliente%TYPE,
  o_nombres_cliente       OUT clientes.nombres%TYPE,
  o_apellidos_cliente     OUT clientes.apellido%TYPE)
IS
BEGIN
  SELECT nombres, apellidos
    INTO o_nombres_cliente, o_apellidos_cliente
    FROM clientes
   WHERE código_cliente = i_codigo_cliente;
END;
```

El mismo código convertido a cursor explicito:

```
PROCEDURE obtener_nombres_cliente (
  i_codigo_cliente   IN     clientes.codigo_cliente%TYPE,
  o_customer_fname      OUT clientes.nombres%TYPE,
  o_customer_lname      OUT clientes.apellidos%TYPE)
IS
  CURSOR l_cur_nombres
  IS
     SELECT nombres, apellidos
       FROM clientes
      WHERE codigo_cliente = i_codigo_cliente;

  l_reg_nombres   l_cur_nombres%ROWTYPE;
BEGIN
  OPEN l_cur_nombres;

  FETCH l_cur_nombres INTO l_reg_nombres;

  CLOSE l_cur_nombres;

  o_nombres_cliente := l_reg_nombres.nombres;
  o_apellido_cliente := l_reg_nombres.apellidos;
END;
```

A primera vista, los cursores explícitos son algo difíciles de entender pues requieren un tanto más de código, sin embargo, el manejo de excepciones se facilita: una excepción TOO_MANY_ROWS no será lanzada cuando se realizan FETCH manual desde un cursor explicito, permitiendo definir las acciones si el caso se presentare, en lugar de manejar excepciones. Esto proporciona al programador una mayor flexibilidad in la estructura del código

El uso de variables del tipo RECORD facilita cambios futuros en las columnas de sentencia SELECT. No es necesario extraer datos en una variable tipo RECORD si se esta ejecutando sentencias como "SELECT 1", consultas COUNT u operaciones similares.

1.8.Sentencias IF

Se debe alinear las sentencias IF con su correspondientes ELSIF, ELSE, y END IF.

IF…ELSIF…ELSE…END IF es la versión PL/SQL's de la sentencia CASE. Si usa una clausula ELSIF en su sentencia IF, debe incluir una cláusula ELSE. Esto permitirá capturar cualquier otro caso que la lógica del programa no tome en cuenta. Aún si piensa que no es posible que exista otra condición, es una Buena práctica de programación capturar cualquier condición con el uso de la clausula ELSE.

Ejemplo:

```
IF l_boolean THEN
   accion;
ELSIF NOT l_boolean THEN
   accion;
END IF;
```

El código anterior trabaja con valores TRUE y FALSE, pero que pasa con el valor NULL?. Es mejor usar (asumiendo que el valor NULL no será tratado de la misma manera como los valores TRUE o FALSE):

```
IF l_boolean THEN
   accion;
ELSIF NOT l_boolean THEN
   accion;
ELSE
   lanzar_excepcion;
END IF;
```

7. Manejo de Errores y Excepciones

La alineación de la palabra clave EXCEPTION será a la izquierda y alineada también con el bloque BEGIN y END que contiene el manejador de excepciones. Identar las demás líneas. Ejemplo:

```
BEGIN
...(varios  sentencias DML)
EXCEPTION
   WHEN e_excepcion_de_usuario THEN
      ROLLBACK;
      escribir_log;
   WHEN OTHERS THEN
      ROLLBACK;
```

```
    RAISE;
END;
```

Considere que una sentencia DML puede necesitar realizar un rollback en caso de problemas. Por tanto, se debe incluir un manejador de excepciones con el manejo de transacciones.

NO use el manejador de excepciones WHEN OTHERS a menos que realmente lo necesite. Normalmente, se requerirá tomar acciones específicas dependiendo del tipo de problema encontrado. WHEN OTHERS está diseñado para manejar excepciones no tratadas.

Los manejadores de excepciones dentro de funciones deberán retornar un valor si la ejecución del programa debe continuar luego de la excepción.

En general, no use manejadores de excepciones para capturar eventos que se supone ocurrirán. En estos casos se recomienda el uso de lógica condicional. (Existen ciertos casos en que la lógica condicional es una mejor alternativa por razones de performance, etc.)

8. Desempeño y Reusabilidad

1.9.SQL del lado del Cliente

No se usará sentencias SQL almacenadas en unidades de programa locales o trigger en el lado del cliente. El lado de cliente reduce el desempeño y la reusabilidad del código.

Excepciones a esta regla incluyen:

Consultas de grupos de registros (Record group queries).

SQL implícito desarrollado por Oracle Forms en conjunción con bloques de tablas de base de datos.

Situaciones que requieran procesar individualmente los registros extraídos desde un cursor en el nivel de forma (por ejemplo, bloques manuales o popularización de grupos de registros (record group population)).

Extracción del siguiente valor de una secuencia de base de datos (u otras extracciones desde la tabla DUAL)

Siempre se ha de ubicar las funciones y procedimientos de base dentro de un paquete. Esto permite un máximo reuso de código y permite también mayor flexibilidad para el DBA al momento de mantener su paquete en el área de memoria compartida en el servidor de base de datos para mejor desempeño.

Asegúrese de que una función o procedimiento no exista en un paquete antes de dedicarse a escribir el código. El administrador del programa será el responsable de administrar los paquetes compartidos.

1.10. Modularidad

El mejor camino para lograr la mínima duplicación de código es promover la modularidad limitando los procedimientos y funciones a tareas individuales. Si su función o procedimiento hace más de una cosa, considere dividirlo en múltiples unidades de programa. Una regla general es que si su procedimiento o función sobrepasa las 3 o 4 páginas, debería ser revisado con el fin de lograr la modularidad en el código.

Si se tiene más de una página de código en un bloque IF..END IF, podría ser dificultoso interpretar dicho código. No es necesario manejar código con bloques IF demasiado grandes. Evite el manejar código de más de una página en estructuras IF y LOOP separando su código en funciones y procedimientos que luego pueden ser invocados dentro de dichas estructuras.

Si se mantienen las unidades de programas pequeñas y con nombres significativos, será mucho más fácil que un nuevo programador que no haya tenido contacto con ese código pueda entenderlo en alto nivel de manera rápida.

1.11. Código duro

Nunca haga referencia a un valor literal dentro de sus programas. Existen muchas alternativas para usar valores literales en el código, incluyendo constantes, variables públicas y privadas a nivel de paquetes, parámetros de procedimientos y funciones, y "constantes" basadas en tablas.

Ejemplo:

Código	Valor Numérico	Valor Cadena	Valor Fecha	DESCRIPCIÓN
ESTADO_CERRADO		CERRADO		Estado que indica si un pedido ha sido completado

En el ejemplo, al hacer referencia a la constante ESTADO_CERRADO, podemos obtener el valor de requerido CERRADO. De esta manera, si en algún momento el valor debe cambiar a COMPLETO, no será necesario realizar ajusten en la programación.

Como se pudo observar, una tabla de constantes comúnmente esta formada por una columna de código y una columna para cada tipo de dato que pueda contener esa constante: numérico, string y fecha; además puede contener una columna de descripción o comentarios.

Las nuevas versiones de Oracle permiten realizar "cast" de los valores almacenados con el tipo de dato varchar2, de tal manera que una tabla de constantes puede reducirse de la siguiente manera:

Código	Valor	DESCRIPCIÓN
ESTADO_CERRADO	CERRADO	Estado que indica si un pedido ha sido completado

Otra manera sutil de código duro es el caso en que se tiene una formula que aparece en el código varias veces. En estos casos es necesario encapsular la lógica repetitiva en una función dentro de un paquete de tal manera que pueda ser invocada desde cualquier punto de la aplicación, por tanto, si la lógica de la fórmula cambia, el cambio en programación se centrará en un solo lugar.

1.12. Funciones de agrupamiento innecesarias

Las funciones de agrupamiento son costosas cuando involucran un gran número de filas. Problemas desempeño se presentan cuando las funciones de agrupamiento son usadas donde realmente no se requiere agrupar datos.

El ejemplo más obvio de este problema lo encontramos con la función COUNT. Hay una regla simple que debe ser recordada para evitar problemas de desempeño relacionados con ésta: No use COUNT a menos que necesite conocer el número exacto de filas que cumplen con su criterio de consulta. Nunca use COUNT si requiere solamente conocer si existe una fila o múltiples filas. El uso de cursores explícitos y de extracciones (fetch) del número de registros que necesite, satisface este requerimiento.

Ejemplo de una consulta COUNT innecesaria: Asumamos que se requiere conocer si existen 0, 1 o muchas filas de tal manera de lanzar una excepción si no existen (zero), usar el dato encontrado (si hay uno), o desplegar una lista que permita escoger al usuario el registro correcto (muchos):

```
DECLARE
  l_nombres clientes.nombres%TYPE;
  l_contador NUMBER := 0;
BEGIN

  l_nombre := 'ROBERTO'

  SELECT NVL(COUNT(1),0)
  INTO   l_contador
  FROM   clientes
  WHERE  nombres = l_nombres;
  IF l_contador = 0 THEN
    desplegar_error;
  ELSIF l_contador = 1 THEN
    usar_dato;
  ELSE
    desplegar_lista;
  END IF;
END;
```

Este procedimiento puede ser escrito de la siguiente manera:

```
DECLARE
  CURSOR l_cur_nombres
  IS
    SELECT 1
    FROM   clientes
    WHERE  name = l_name;
  l_nombres clientes.nombres%TYPE;
  l_encontrado NUMBER := 0;
BEGIN
  l_nombres := 'ROBERTO';
  OPEN l_cur_nombres;
  LOOP
    FETCH l_cur_nombres INTO l_encontrado;
    EXIT WHEN l_cur_nombres%NOTFOUND OR l_cur_nombres%ROWCOUNT > 1;
  END LOOP;
```

```
  IF l_cur_nombres%ROWCOUNT = 0 THEN
    CLOSE l_cur_nombres;
    desplegar_error;
  ELSIF l_cur_nombres%ROWCOUNT = 1 THEN
    CLOSE l_cur_nombres;
    usar_dato;
  ELSE
    CLOSE l_cur_nombres;
    desplegar_lista;
  END IF;
END;
```

9. Estándares de nombrado

1.13. Base de Datos

1.13.1. Objetos de BDD

La siguiente lista de reglas se aplica a los identificadores:

- Longitud de 1 a 30 bytes con las siguientes excepciones:
- Nombres de bases de datos están limitadas a 8 bytes.
- Nombres de “database links” pueden tener una longitud de 128 bytes.
- Deben ser diferentes a las palabras reservadas de Oracle Database.

Objeto	Descripción	Ejemplo
Tablespaces	Nombre significativo. Se recomienda usar un tablespace para datos y otro para índices	comercial_dat comercial_idx
Data Files	Nombres tablespace Sufijo _DAT para tablespaces de datos, _IDX para el caso de tablespaces designado para almacenar índices	comercial_dat_01.dbf comercial_idx_01.dbf
Directorios	Nombre directorio Sufijo _DIR	ARCHIVOS_DIR
Usuarios	Usuarios deben tener relación con los nombres de usuarios registrados en e l domino REDEMELNORTE	elopez
Tablas	Nombre largo Prefijo T Nombre tabla en plural	tclientes
	Nombre corto Prefijo T Nombre corto de la tabla (3 caracteres)	tcomcli
Vistas	Prefijo V Nombre de la vista	vclientes
Índices	Nombre columna (nombre del índice) Sufijo _IDX Los índices correspondientes a claves primarias deberán contener el sufijo _PK	codigo_cliente_idx
Secuencias	Prefijo seq Nombre secuencia (Nombre de tabla donde es usada)	Seq_numero_orden
Columnas	Nombre de columna	
Claves Primarias	Nombre de clave Sufijo _PK	orden_compra_pk

Objeto	Descripción	Ejemplo
Claves Foráneas	Prefijo (nombre corto) de la tabla a la que se hace referencia Nombre de columna Sufijo _FK	cli_codiigo_cliente_fk
Check	Nombre de chexk constraint Sufijo _CK	cedula_nula_ck
Procedimientos	Prefijo PRO_ Nombre procedimiento	pro_calcular_rol
Funciones	Prefijo FUN_ Nombre procedimiento	fun_cedula_valida
Paquetes	Prefijo módulo Identificador de paquete _PCK_ Nombre paquete	com_pck_clientes
Triggers	Prefijo TRG_ Abreviatura acción (A)fter, (B)efore, (I)nsert, (U)pdate, (D)elete Nombre trigger	orden_compra_aiu acceso_bd
Jobs	Prefijo JOB	job_actualizar_cartera
Sinónimos	En el mayor de los casos se usa el mismo nombre del objeto al cual el sinónimo hace referencia	
DB Links	Prefijo DBL_ Base de datos a la que hace referencia	dbl_alpha
Vistas Materializadas	Prefijo VM_ Identificador módulo (3 caracteres) Nombre de la vista materializada	vm_com_clientes
Roles	Prefijo ROL_ Identificador de módulo Nombre del rol	com_rol_cajero

1.13.2. Información de auditoría

En general, las tablas que involucren:

- Parametrización
- Transacciones de la aplicación
- Seguridades

Deberán incluir siempre las siguientes columnas de auditoría:

- USUARIO_CREACION
- FECHA_CREACION
- USUARIO_MODIFICACION
- FECHA_MODIFICACION

1.1.1.4. Opción de "Journal tables" de Oracle Designer.

Al tener el modelo de Base de Datos en la herramienta Oracle Designer, existe la opción de usar las siguientes características:

Journal Tables

Una tabla journal permite almacenar el detalle sobre cada insert, update o delete realizados en la tabla asociada.

Table API

Un conjunto de objetos del lado del servidor (triggers y packages) diseñados para asegurara las reglas de datos en el servidor independientemente del tipo de aplicación cliente

1.13.3. Glosario de prefijos

Prefijo		Significado
Módulos/Sistemas		
	GEN_	GENERAL. catálogos generales que pueden ser comunes a los diferentes módulos
	COM	COMERCIAL
	PRE_	PRESUPUESTO
	CON_	CONTABILIDAD
	ADQ_	ADQUISICIONES
	CFG_	CONFIGURACIONES
	NBX_	Sistema NBX
	NOM_	NOMINAS
	FAC_	FACTURACION
	INV_	INVENTARIOS
Abreviaturas		
	CCO	Centros de Costo
	LOC	Localizaciones
	ARE	Areas

Prefijo		Significado
Esquemas (Usuarios propietarios de los diferentes módulos o sistemas)		
	SIGORG	Sistema de gestión de la organización
	SIGCOM	Sistema comercial
	SIGPRE	PRESUPUESTO
	SIGCON	CONTABILIDAD
	SIGADQ	ADQUISICIONES
	SIGCFG	CONFIGURACIONES
	NBX	
	SIGNOM	NOMINAS
	SIGFAC	FACTURACION
	SIGINV	INVENTARIOS
	SIGPRO	PROYECTOS
Abreviaturas		
	CCO	Centros de Costo
	LOC	Localizaciones
	ARE	Areas