




	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	
	06. Instrucciones Técnicas	
	06.02. Normas de Construcción PL/SQL	

Índice

1.	Introducción	3
2.	Nomenclatura.....	3
3.	Conceptos Elementales.....	4
3.1.	Facilitar la comprensión del código	4
3.1.1.	Codificar sin aplicar ideas complejas	4
3.2.	Prevenir la duplicidad de código.....	5
4.	Tratamiento de la Información.....	6
4.1.	Cursores	6
4.1.1.	Apertura y lectura de Cursores	6
4.1.2.	Evitar cursores implícitos.....	6
4.1.3.	Cierre de cursores.	7
4.2.	SQL Dinámico.....	10
4.2.1.	DMBS_SQL frente a Execute Immediate.....	10
5.	Flujo de la información	12
5.1.1.	No es conveniente utilizar la sentencia GOTO.....	12
5.1.2.	Las excepciones no deben enmascarar la sentencia <i>goto</i>	12
5.1.3.	Evitar el uso de exit y return de forma no estructurada.....	13
5.1.4.	No usar demasiados if anidados	13
5.1.5.	Racionalizar el uso de bloques internos.....	15
5.1.6.	Expresiones estáticas dentro de bucles	16
6.	Estructuras Condicionales	17
6.1.	Comparaciones con cadenas vacías	17
6.2.	NOT: Uso inadecuado.....	18
6.3.	Optimización de expresiones condicionales	19
6.4.	Optimización del flujo de las sentencias condicionales	20
7.	Parámetros	21
7.1.	Especificar siempre el modo del parámetro.....	21
7.2.	Usar siempre el mismo orden en los parámetros.....	22
7.3.	Evitar restricciones not null	23
7.4.	No utilizar notación nominal.....	24
7.5.	NOCOPY	25
8.	Variables y Tipos de datos	26
8.1.	Elección del tipo de dato	26
8.2.	No abusar de las variables globales.....	27
8.3.	Constantes SIRhUS.....	28
8.3.1.	Uso de constantes.	28
8.3.2.	Gestión de constantes.....	28
8.4.	Eventos.	29
9.	Manipulando fechas.....	30

	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	
	06. Instrucciones Técnicas	
	06.02. Normas de Construcción PL/SQL	

9.1. Comparación de Fechas.	30
9.2. Truncamiento de Fechas	31
9.3. Función LAST_DAY.....	32
9.4. Precaución en el uso del SYSDATE.....	33
10. Control de Excepciones	34
10.1. Excepciones en la sección declarativa	35
10.2. Propagación de Excepciones	36
10.2.1. Uso de Raise_Application_Error con parámetros.....	37
10.3. When Others.....	38
10.4. Pila de error y pila de llamadas	40
10.5. Cerrar estructuras dinámicas de datos en Exception	42
10.6. Gestión de errores en SIRhUS.	43
10.6.1. RAISE_APPLICATION_ERROR en SIRhUS	45
10.6.2. IH_PR_OTHERS_EXCEPTION	46
10.6.3. IH_PR_MSG_OTHER_EXCP	48
10.6.4. IH_PR_TEXTO_ERROR	49
10.6.5. Normalización de los códigos de error.	50
11. Consultas Prácticas.....	51
11.1. Solapamiento de fechas.....	51
11.2. Cálculo de los años, meses y días transcurridos entre dos fechas.	53
11.3. Calculo de los días laborables incluidos entre dos fechas.	54

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Instrucciones Técnicas	
	06.02. Normas de Construcción PL/SQL	

1. Introducción



En este documento se recogen Normas y Recomendaciones de uso general que se deben aplicar en la Construcción de componentes en lenguaje PL/SQL. Estas normas y recomendaciones son válidas tanto en la construcción de componentes de base de datos, formularios Developer, Informes Developer, así como otros componentes que utilicen PL/SQL para su desarrollo. Este documento no es por tanto un manual de PL/SQL, únicamente pretende poner de manifiesto aquellos puntos que la experiencia aconseja tener en cuenta para un correcto desarrollo. Este documento presupone un conocimiento, al menos básico, del lenguaje PL/SQL.

2. Nomenclatura.

Todos los objetos de Base de Datos, así como los identificadores de tipos y variables, están precedidos de un prefijo que informa del tipo de objeto al que representa. Además, para el caso de los objetos de bse de Datos, por delante de estos prefijos aparece el prefijo de la aplicación IH_.

Tipo de Objeto	Prefijo	Longitud máxima
Procedimientos	IH_PR_	15
Paquetes	IH_PQ_	15
Funciones	IH_FU_	15
Cursores	CU_	15
Parámetros	P_	
Variables	V_	
Variables Globales	VG_	
Variables tipo registro (rowtype)	R_	
Definiciones de tipo	T_	

En el caso de Procedimientos y Funciones definidas en paquetes se puede omitir el prefijo de la aplicación.

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Instrucciones Técnicas	
	06.02. Normas de Construcción PL/SQL	

3. Conceptos Elementales

3.1. Facilitar la comprensión del código



Se debe intentar que la estructura del programa o las sentencias, no compliquen la comprensión de la finalidad que persigue el código.

Expresiones complejas en las llamadas a procedimientos o funciones, expresiones condicionales complejas, etc. se deben simplificar. En ocasiones el uso de variables temporales facilita la comprensión de sentencias que de otra forma serían difíciles de entender.



Solución incorrecta

```
-- Calcular la edad de una persona a partir de una fecha base
FUNCTION fcaledde
  (p_base          IN VARCHAR2
  ,p_nacimiento    IN VARCHAR2)
RETURN NUMBER IS
BEGIN
  RETURN(FLOOR(FLOOR(MONTHS_BETWEEN(TO_DATE(p_base,
'DD/MM/YYYY'), TO_DATE(p_nacimiento, 'DD/MM/YYYY')))/12));
END fcaledde;
```



Solución correcta

```
-- Calcular la edad de una persona a partir de una fecha base
FUNCTION fcaledde
  (p_base          IN VARCHAR2
  ,p_nacimiento    IN VARCHAR2)
RETURN NUMBER IS
  v_meses NUMBER;
  v_años   NUMBER;
BEGIN
  v_meses := FLOOR(MONTHS_BETWEEN(TO_DATE(p_base,
'DD/MM/YYYY'), TO_DATE(p_nacimiento,
'DD/MM/YYYY')));
  v_años := FLOOR(v_meses)/12;



  RETURN(v_años);
END fcaledde;
```

3.1.1. Codificar sin aplicar ideas complejas

La incorporación de código complejo que resuelve un problema de una manera artificial implica un código poco legible o difícilmente entendible, tanto por los demás desarrolladores como por el propio autor a lo largo del tiempo.



En aquellas ocasiones en que un código difícil de entender permita mejorar ostensiblemente el rendimiento de un programa, o simplificar de manera extraordinaria la extensión del código, se podrá utilizar estas prácticas de codificación. En estos casos, se deberá incluir comentarios detallados de los fundamentos de la idea utilizada, para ayudar al seguimiento del código y su comprensión.

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Instrucciones Técnicas	
	06.02. Normas de Construcción PL/SQL	

3.2. Prevenir la duplicidad de código



No se debe repetir código a lo largo de la aplicación. Cuando dos porciones de código con la misma funcionalidad están en sitios distintos, es posible que deba existir un procedimiento o función con dicha funcionalidad.

No siempre un código que se repite es candidato a formar un procedimiento o función. Normalmente este tipo de código tiene las siguientes características:

- Se repite en la aplicación.
- Podría ser parametrizable.
- No depende su funcionalidad de su ámbito.
- Si la funcionalidad varía, variará en todos los sitios dónde aparezca ese código.



El código que cumple con estas condiciones debería formar una función, procedimiento o unidad de programa para facilitar su mantenimiento.



```

Ejemplo correcto
FUNCTION fvalenif
  (p_nif IN VARCHAR2 )
RETURN NUMBER IS
  v_novale VARCHAR2(20);
  e_error EXCEPTION;
BEGIN
  IF p_nif IS NULL
    OR NVL(LTRIM(p_nif,' '), ' ') = ' ' THEN
    RAISE e_error;
  END IF;
  --
  SELECT '1'
    INTO v_novale
   FROM SU_CLIENT
   WHERE NIFCLIEN = p_nif;
  RETURN(0);
EXCEPTION
  --
  WHEN e_error THEN
    RAISE_APPLICATION_ERROR (-20000
                              , 'SUR-00951 #1 FVALENIF');
  WHEN NO_DATA_FOUND THEN
    RETURN(1);
  WHEN TOO_MANY_ROWS THEN
    RETURN(2);
  WHEN OTHERS THEN
    RAISE_APPLICATION_ERROR(-20000
                              , 'SUR-02000 #1'||SQLERRM
                              || ' #2 FVALENIF');
END fvalenif;

```

 JUNTA DE ANDALUCÍA CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Instrucciones Técnicas 06.02. Normas de Construcción PL/SQL	

4. Tratamiento de la Información

4.1. Cursores

4.1.1. Apertura y lectura de Cursores



Para recorrer un cursor la forma recomendada es mediante la fórmula **For ... Loop**

De las formas existentes de abrir y recorrer un cursor es recomendable la utilización de la fórmula **For ... Loop** frente a **Open ... Loop ... Fetch**. De esta forma se requieren menos líneas de código.



Cuando se precise recuperar un único valor del cursor utilizaremos la fórmula **Open ... Fetch**, sin bucle.



4.1.2. Evitar cursores implícitos



Utilizar cursores explícitos en lugar de cursores implícitos.

Las ventajas de utilizar cursores explícitos son varias frente a la fórmula **select ... into** (cursor implícito):

- La fórmula **select ... into** requiere el doble de memoria que un curso explícito al definir un cursor implícito.
- El uso de cursores explícitos reduce la posibilidad de aparición de excepciones del tipo **no_data_found** o **too_many_rows**.

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Instrucciones Técnicas	
	06.02. Normas de Construcción PL/SQL	



EJEMPLO: Uso incorrecto de cursor implícito

```

DECLARE
  V NUMBER(8);
BEGIN

  SELECT X_PERSONA
    INTO V
    FROM IHPERSONAS
    WHERE N_IEP = '28567431';

EXCEPTION
  WHEN NO_DATA_FOUND THEN
    ..

END;
```



EJEMPLO: Uso correcto mediante uso de cursor

```

DECLARE

  CURSOR C_EJ IS
    SELECT X_PERSONA
      FROM IHPERSONAS
      WHERE N_IEP = '28567431';

  V NUMBER(8);
BEGIN

  OPEN C_EJ;
  FETCH C_EJ INTO V;
  IF C_EJ%NOTFOUND THEN
    ...
  END IF;
  CLOSE C_EJ;

END;
```



4.1.3. Cierre de cursores.



Evitar tener cursores abiertos innecesariamente

Con objeto de evitar en lo posible exceder el parámetro *maxopencursors* (máximo número de cursores abiertos por una sesión de usuario en base de datos) se deben de cerrar los cursores que ya no se necesiten antes de abrir otros cursores.

Cuando se usa un cursor para recuperar los valores y/o comprobar la existencia de una única fila, éste tiene que cerrarse inmediatamente después de su lectura o, en su caso, de la comprobación de la existencia o inexistencia de valores.

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Instrucciones Técnicas	
	06.02. Normas de Construcción PL/SQL	

Casos incorrectos:



EJEMPLO: El cursor se mantiene abierto innecesariamente durante la ejecución del procedimiento IH_PR_PROCESO

```

DECLARE
CURSOR CU_TEST( P_CODIGO ) IS
  SELECT L_TIPO
    FROM IHTABLA
   WHERE C_CODIGO= P_CODIGO;
  V_AUX IHTABLA.L_TIPO%TYPE;
BEGIN
  OPEN CU_TEST(IH_FU_VALOR('CTAR'));
  FETCH CU_TEST INTO C_AUX;
  IH_PR_PROCESO(V_AUX);
  CLOSE CU_TEST;
EXCEPTION
<< manejo de la excp.>>
END;
```



EJEMPLO: El cursor se mantiene abierto y se abre otro cursor.

```

DECLARE
  CURSOR CU_1 IS ...;
  CURSOR CU_2 IS ..
BEGIN
  OPEN CU_1;
  FETCH CU_1 INTO ...
  IF CU_1%FOUND THEN
    OPEN CU_2;
    <...>
    CLOSE CU_2;
  END IF;
  CLOSE CU_1;
EXCEPTION
<< manejo de la excp.>>
END;
```



Casos correctos:



EJEMPLO: El cursor se cierra antes de llamar al procedimiento.

```

DECLARE
CURSOR CU_TEST( P_CODIGO ) IS
  SELECT L_TIPO
    FROM IHTABLA
   WHERE C_CODIGO= P_CODIGO;
  V_AUX IHTABLA.L_TIPO%TYPE;
BEGIN
  OPEN CU_TEST(IH_FU_VALOR('CTAR'));
  FETCH CU_TEST INTO C_AUX;
  CLOSE CU_TEST;
  IH_PR_PROCESO(V_AUX);
EXCEPTION
<< manejo de la excp.>>
END;
```


 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Instrucciones Técnicas	
	06.02. Normas de Construcción PL/SQL	



EJEMPLO: El resultado %found / %notfound se almacena en una variable booleana.



```
DECLARE
CURSOR CU_1 IS ...;
CURSOR CU_2 IS ..
V_EXISTE BOOLEAN;
BEGIN
OPEN CU_1;
FETCH CU_1 INTO ...
V_EXISTE := CU_1%FOUND; -- o %NOTFOUND
CLOSE CU_1;

IF V_EXISTE THEN
OPEN CU_2;
<...>
CLOSE CU_2;
END IF;
EXCEPTION
<< manejo de la excp.>>
END;
```



EJEMPLO: Se cierra antes de seguir. (Este código es correcto pero es preferible el ejemplo anterior).

```
DECLARE
CURSOR CU_1 IS ...;
CURSOR CU_2 IS ..
BEGIN
OPEN CU_1;
FETCH CU_1 INTO ...
IF CU_1%FOUND THEN
CLOSE CU_1;
OPEN CU_2;
<...>
CLOSE CU_2;
ELSE
CLOSE CU_1;
END IF;
EXCEPTION
<< manejo de la excp.>>
END;
```

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Instrucciones Técnicas	
	06.02. Normas de Construcción PL/SQL	

4.2. SQL Dinámico

4.2.1. DMBS_SQL frente a Execute Immediate



Cuando haya posibilidades de elección, es preferible el uso de la instrucción **Execute Immediate** frente al paquete DMBS_SQL

El uso de **Execute Immediate** no requiere de tantas instrucciones como el del paquete DMBS_SQL. El código resultante es más simple.



En la construcción de pl/sql dinámico es obligatorio que los valores se pasen mediante bind-variables

Limitaciones en el uso de *Execute Immediate*:



- Sólo se puede usar dentro de un objeto base de datos.
- No aplicable sobre cursores, siendo válido únicamente para sentencias *select* que devuelven una sola fila.



EJEMPLO: Usos de SQL Dinámico

```
DECLARE
  V_VALOR1 NUMBER(10) := ...;
  V_VALOR2 NUMBER(10) := ...;
  V_PROCESO VARCHAR2(30) := 'IH_PR_COHTXT';
BEGIN
  EXECUTE IMMEDIATE
    'BEGIN ` || V_PROCESO || ` (:P1,:P2); END;' USING V_VALOR1, V_VALOR2;
END;
```

```
DECLARE
  V_SQL VARCHAR2(2000) := ' SELECT N_IEP FROM IHPERSONAS WHERE
X_PERSONA = :XPER';
  V_N_IEP IHPERSONAS.N_IEP%TYPE;
BEGIN
  EXECUTE IMMEDIATE V_SQL INTO V_N_IEP USING 110;
  DBMS_OUTPUT.PUT_LINE(V_N_IEP);
END;
```

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Instrucciones Técnicas	
	06.02. Normas de Construcción PL/SQL	



```

DECLARE
    CURSOR C_DATSOL IS
        SELECT X_DATSOL, V_TIPO, FU.D_PAQUETE, FU.S_FUNCIONESFAS, S_TIPDAT
        FROM IHCABSOL CS, IHDATSOL DS, IHTIPDAT TD, IHFUNCIONESFAS FU
        WHERE CS.X_CABSOL = P_X_CABSOL
        AND CS.CASI_X_CABSOL IS NULL
        AND CS.EJERCICIO = DS.EJERCICIO
        AND CS.AYUF_X_AYUDASFAS = DS.AYUF_X_AYUDASFAS
        AND ( DS.V_AMBITO IN ('S', 'A', 'C')
        OR DS.V_AMBITO IS NULL)
        AND DS.TIDA_X_TIPDAT = TD.X_TIPDAT
        AND DS.FUNC_X_FUNCIONESFAS = FU.X_FUNCIONESFAS;

    V_BLOQUE    VARCHAR2 (500);
BEGIN
    /*Borramos tabla temporal de Avisos*/
    IH_PQ_VALIDACION.IH_PR_BORRAR_AVISOS;



    FOR R_DATSOL IN C_DATSOL LOOP
        /*Realiza la llamada dinámicamente de la función o procedimiento
        relacionado con el x_datsol*/
        V_BLOQUE :=
            'BEGIN '
            || R_DATSOL.D_PAQUETE || '.' || R_DATSOL.S_FUNCIONESFAS
            || '(:P_CABSOL,:P_DATSOL,:P_V_TIPO,:P_S_TIPDAT); END;';
        EXECUTE IMMEDIATE V_BLOQUE
            USING P_X_CABSOL, R_DATSOL.X_DATSOL, R_DATSOL.V_TIPO,
R_DATSOL.S_TIPDAT;
    END LOOP;
EXCEPTION
    WHEN OTHERS THEN
        RAISE_APPLICATION_ERROR (-20000, 'APL-60027#1
IH_PQ_FUNTIPDAT.IH_PR_CALCULADATSOL');
END;

```



En SIRhUS se aplica SQL dinámico en, entre otros:

- SirhusA: Ejecución de validaciones y eventos
- SirhusF: Funciones de cálculo de baremo
- SirhusG: Funciones de validación de solicitudes ...

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Instrucciones Técnicas	
	06.02. Normas de Construcción PL/SQL	

5. Flujo de la información

5.1.1. No es conveniente utilizar la sentencia GOTO



El uso de la sentencia *GOTO* no es recomendable en una programación estructurada



Solución incorrecta

```

IF condicion THEN
    GOTO etiqueta; -- Salta a la etiqueta
END IF;
...
<<etiqueta>>
... -- El control continúa aquí
...

```

5.1.2. Las excepciones no deben enmascarar la sentencia *goto*



No se debe utilizar las excepciones para simular saltos no estructurados como con una sentencia *GOTO*. Se deben utilizar las excepciones sólo y exclusivamente para el tratamiento de errores.

No se deben usar las excepciones para simular saltos no estructurados, ni para enmascarar salidas mediante *RETURN* que den como resultado un código no estructurado.



Solución incorrecta

```

IF condicion THEN
    RAISE e_miExcepcion; -- No es un error, enmascara un GOTO
END IF;
...
EXCEPTION
    WHEN e_miExcepcion THEN
        ... -- Código de opción, no de error
END;

```





Solución correcta

```

IF condicion THEN
    ... -- Código de opción, no de error
END IF;
...
EXCEPTION
    ... -- Tratamiento de errores
END LOOP;

```

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Instrucciones Técnicas	
	06.02. Normas de Construcción PL/SQL	

5.1.3. Evitar el uso de `exit` y `return` de forma no estructurada



Para facilitar la lectura y seguimiento del código se debe evitar un código no estructurado de `EXIT` o `RETURN`.

- No se debe salir de bucles `FOR ... LOOP` mediante `EXIT` o `RETURN`.
- No se debe usar la sintaxis de `EXIT` en bucles `WHILE`.
- Se debe tener una única salida mediante `RETURN` en el código de una función en un funcionamiento en el que no se produzcan excepciones.



Solución incorrecta

```
FOR ...
LOOP
    ...
    IF condicion THEN
        EXIT;
    END IF;
END LOOP;
```



Solución correcta

```
LOOP
    ...
    EXIT WHEN condicion
END LOOP;
```

5.1.4. No usar demasiados `if` anidados





No se debe hacer un uso elevado de sentencias `IF` anidadas, pues ello provoca que el código sea de difícil lectura y siempre dificulta el mantenimiento.

Preferiblemente no se debe alcanzar un nivel de anidamiento superior a 3. Si la lógica requiere un nivel de anidamiento mayor, quizás hay que volver a plantear el algoritmo o modularizar más el componente desarrollado. Es más recomendable hacer uso de la instrucción `ELSIF`





Solución incorrecta

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Instrucciones Técnicas	
	06.02. Normas de Construcción PL/SQL	

```

IF condicion1 THEN
    ...
    IF condicion2 THEN
        ...
        IF condicion3 THEN
            ...
            IF condicion4 THEN
                ...
                IF condicion5 THEN
                    ...
                    END IF;
                ...
            END IF;
            ...
        END IF;
        ...
    END IF;
    ...
END IF;
    ...
END IF;

```

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Instrucciones Técnicas	
	06.02. Normas de Construcción PL/SQL	

5.1.5. Racionalizar el uso de bloques internos



No se debe hacer un uso inadecuado de los bloques internos.

La posibilidad de utilización de bloques internos es proporcionada por PL/SQL con el objetivo de controlar las excepciones del código incluido en dicho bloque.

No se deben crear bloques internos para incorporar diferentes bloques lógicos de código, ni para diferenciar grupos de sentencias diferentes del código desarrollado, si no es necesario por el tratamiento de excepciones.



Solución incorrecta.

```
BEGIN
  ...
  -- Cálculo de la base imponible
  BEGIN
    <código del cálculo de la base imponible>
  END;
  ...
END;
```





Solución Correcta, manteniendo el código

```
BEGIN
  ...
  -- Cálculo de la base imponible
  <código del cálculo de la base imponible>
  ...
END;
```



Solución Correcta, mediante funciones/procedimientos

```
BEGIN
  ...
  -- Cálculo de la base imponible
  v_base_imponible := <función cálculo de base imponible>
  ...
END;
```



	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	
	06. Instrucciones Técnicas	
	06.02. Normas de Construcción PL/SQL	

5.1.6. Expresiones estáticas dentro de bucles



No situar expresiones estáticas dentro de bucles

Por pura lógica es innecesario ejecutar repetidamente partes de código invariables. Basta con situarlos antes de la apertura del bucle.

	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	
	06. Instrucciones Técnicas	
	06.02. Normas de Construcción PL/SQL	

6. Estructuras Condicionales

6.1. Comparaciones con cadenas vacías

El realizar comparaciones entre variables o datos hay que tener cuidado que ninguno de los operandos se corresponda a un valor nulo, ya que en este caso el resultado de la operación será invariablemente FALSE, independientemente del tipo de comparación, lo que puede provocar resultados no deseados. Obsérvese la siguiente tabla de valores:



Comparación	Resultado
(NULL = algo)	FALSE
(NULL <> algo)	FALSE
(NULL = NULL)	FALSE
(NULL <> NULL)	FALSE



EJEMPLO: Condición que nunca se cumple

```

DECLARE
    CTE    CONSTANT VARCHAR2 (5) := IH_FU_VALOR
    ('TIAN');
    VL_C_CODIGO    VARCHAR2 (2) := NULL;
BEGIN
    IF NVL (VL_C_CODIGO, '') <> CTE THEN
        DBMS_OUTPUT.PUT_LINE ('dentro del if');
    END IF;
END;
```

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Instrucciones Técnicas	
	06.02. Normas de Construcción PL/SQL	

6.2. NOT: Uso inadecuado.





Nunca se debe utilizar el operador **NOT** para obtener el resultado contrario a un operador de comparación.

NOT en la mayoría de sus usos inutiliza los índices.

- **Alternativas**

El comportamiento del operador **NOT** puede emularse con el uso del resto de operadores lógicos ('<', '<=', '>', '>=').

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Instrucciones Técnicas	
	06.02. Normas de Construcción PL/SQL	

6.3. Optimización de expresiones condicionales



Un orden adecuado en las condiciones de las sentencias de control puede hacer que no se evalúen condiciones costosas de manera innecesaria, acelerando la aplicación *PL/SQL*.

La evaluación de expresiones lógicas compuestas en **PL/SQL** finaliza en cuanto que un resultado no verifica las condiciones requeridas, por tanto, es aconsejable la colocación de las expresiones lógicas más costosas en último lugar por evaluarse de izquierda a derecha las expresiones lógicas.





De esta forma se tiene que llamar a la función siempre

```
IF tieneVacaciones(v_empleadoid) AND (v_sueldo < 1300) THEN
...
END IF;
```



De esta forma se llama a la función sólo si se cumple la primera condición

```
IF (v_sueldo < 1300) AND tieneVacaciones(v_empleadoid) THEN
...
END IF;
```

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Instrucciones Técnicas	
	06.02. Normas de Construcción PL/SQL	

6.4. Optimización del flujo de las sentencias condicionales



En las sentencias condicionales es recomendable la colocación de las ramas cuyas condiciones se cumplen con mayor frecuencia al principio, para conseguir una mejora en el rendimiento.

Dado que la mejora en el rendimiento, en la mayoría de los casos no será muy apreciable, si la colocación de las ramas de la estructura condicional implica una menor claridad de código, se optará por ubicarlas dando prioridad a la comprensión del código resultante.





Se evalúan todas las condiciones casi siempre

```
IF sucede_1% THEN
...
ELSIF sucede_2% THEN
...
ELSIF sucede_3% THEN
...
ELSIF sucede_4% THEN
...
ELSE -- 90% de las ocasiones
...
END IF;
```



La mayoría de las veces se evalúa una condición

```
IF sucede_90% THEN
...
ELSIF sucede_4% THEN
...
ELSIF sucede_3% THEN
...
ELSIF sucede_2% THEN
...
ELSE -- 1% de las ocasiones
...
END IF;
```

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Instrucciones Técnicas	
	06.02. Normas de Construcción PL/SQL	

7. Parámetros

7.1. Especificar siempre el modo del parámetro



Siempre se deben especificar si los parámetros son de entrada (IN), salida (OUT) entrada / salida (IN OUT).

Identificar el tipo de parámetro para aquellos que son sólo de entrada (siendo ésta la opción por defecto) mejora la lectura del código.





Solución incorrecta

```
FUNCTION fcaledde
  (p_base      VARCHAR2
  ,p_nacimiento VARCHAR2)
...
```



Solución correcta

```
FUNCTION fcaledde
  (p_base      IN VARCHAR2
  ,p_nacimiento IN VARCHAR2)
...
```

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Instrucciones Técnicas	
	06.02. Normas de Construcción PL/SQL	

7.2. Usar siempre el mismo orden en los parámetros



Es recomendable especificar los tipos de parámetros en un mismo orden. El orden debe ser: Entrada (IN), modificables (IN OUT), salida (OUT), parámetros por defecto.

Utilizar el mismo orden de los tipos de parámetros para todos los módulos facilita la lectura de código y su mantenimiento. Si se colocan los parámetros por defecto al final, se puede utilizar siempre la notación posicional, sin tener que recurrir a la notación nominal.



En casos Excepcionales en los que los parámetros formen parte de una estructura (más o menos compleja) de un objeto y tenga una gran relevancia el orden de las estructuras de las tablas o registros usados, se puede mantener el orden lógico de esa estructura para la interfaz de todas las unidades de programa sin respetar el orden: Entrada, Entrada/Salida y Salida.



Solución incorrecta

```
FUNCTION F040901
(p_codconmq OUT VARCHAR2
,p_coddeleg OUT VARCHAR2
,matricula IN VARCHAR2
,guia IN VARCHAR2
,p_numidfau OUT VARCHAR2
,rastreo IN VARCHAR2 DEFAULT 'N')
...
```





Solución correcta

```
FUNCTION F040901
(p_matricula IN VARCHAR2
,p_guía IN VARCHAR2
,p_codconmq OUT VARCHAR2
,p_coddeleg OUT VARCHAR2
,p_numidfau OUT VARCHAR2
,p_rastreo IN VARCHAR2 DEFAULT 'N')
...
```



No se debe modificar el orden de las funciones o procedimientos ya existentes y que sean utilizados de forma pública (por diferentes componentes software). En caso de que se requiera una nueva ordenación, se deberán tomar las precauciones necesarias para que todas sus llamadas sean acondicionadas a este nuevo orden.

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Instrucciones Técnicas	
	06.02. Normas de Construcción PL/SQL	

7.3. Evitar restricciones not null



En **PL/SQL** el uso de restricciones (**CONSTRAINTS**) *NOT NULL* en la declaración de variables, reduce la eficiencia.



Ejemplo con NOT NULL como CONSTRAINT

```

PROCEDURE SumaSimple IS
  m NUMBER NOT NULL :=1;
  a NUMBER;
  b NUMBER;
BEGIN
  ...
  m := a + b; -- Excepción VALUE_ERROR si a ó b es NULL
  ...
EXCEPTION
  WHEN VALUE_ERROR THEN
    <código para m nulo>
END;
```



En el ejemplo anterior, el valor de $a + b$ se asigna a una variable temporal. Se comprueba si el valor es nulo o no. Si no es nulo se asigna a m. Si es nulo se eleva una excepción. En el siguiente ejemplo se puede observar como simular el comportamiento de *NOT NULL* de manera más eficiente en código.



Ejemplo con testeo de NOT NULL en código

```

PROCEDURE SumaSimple IS
  m NUMBER; <sin constraint>
  a NUMBER;
  b NUMBER;
BEGIN
  ...
  m := a + b;
  IF m IS NULL THEN
    <código para m nulo>
  ...
END;
```

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Instrucciones Técnicas	
	06.02. Normas de Construcción PL/SQL	

7.4. No utilizar notación nominal



No se debe usar la notación nominal, para evitar efectos colaterales en las llamadas a funciones o procedimientos externos, debidos a cambios en los nombres de sus parámetros por el responsable de dicha unidad de programa externa.



Solamente se debe utilizar la notación nominal cuando se tenga que realizar una llamada a una función o procedimiento con parámetros opcionales sin usar alguno de ellos que no sea sólo el último parámetro.



INCORRECTO

```

PROCEDURE ejemplo_1
  (p_fijo      IN VARCHAR2
  ,p_opcional_1 IN NUMBER DEFAULT 1
  ,p_opcional_2 IN NUMBER DEFAULT 2);
...
PROCEDURE ejemplo_2
  (p_fijo      IN VARCHAR2
  ,p_parametro_1 IN NUMBER
  ,p_parametro_2 IN NUMBER);
...
ejemplo_1(p_fijo      => v_fijo,
          p_opcional_1 => 2,
          p_opcional_2 => 1);
...
ejemplo_2(p_fijo      => v_fijo,
          p_parametro_1 => 2,
          p_parametro_2 => 1);
...

```



CORRECTO



```

...
ejemplo_1(v_fijo, p_opcional_2 => 1);
...
ejemplo_1(v_fijo, 2, 1);
...

```



Se debe tener especial cuidado al cambiar los nombres de los parámetros opcionales de una función o procedimiento ya que puede provocar fallos en unidades de programas externas que usen la notación nominal de forma correcta.

	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	
	06. Instrucciones Técnicas	
	06.02. Normas de Construcción PL/SQL	

7.5. NOCOPY



Es conveniente utilizar NOCOPY para los parámetros de tipo OUT o IN OUT



Cuando pasamos argumentos a través de una lista de parámetros, estos argumentos pueden pasarse por referencia o por valor:

- *Por referencia* la estructura de datos dentro del procedimiento apunta a la misma posición de memoria que el del argumento.
- *Por valor* el valor del argumento es copiado en estructura de datos del programa.

Por defecto el paso de parámetros a funciones o procedimientos sigue la siguiente regla:

- Los parámetros IN son pasados por referencia
- Los parámetros OUT o IN OUT son pasados por valor

Cuando pasamos una estructura de datos grande (un registro por ejemplo) puede producirse una degradación en la memoria y en el funcionamiento de la aplicación.

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Instrucciones Técnicas	
	06.02. Normas de Construcción PL/SQL	

8. Variables y Tipos de datos



8.1. Elección del tipo de dato



Definir las variables con el el tipo de dato más adecuado

Aparte de los tipos de datos habitualmente utilizados (**DATE**, **VARCHAR2** y **NUMBER**), existen por supuesto otros muchos. Debe elegirse el más adecuado al tamaño y precisión del tipo de información que albergará. Si por evitar problemas elegimos tipos de datos desmesurados, estaremos malgastando memoria.

Tipo de dato	Comentario
NUMBER	Si no se especifica la precisión Oracle reserva espacio para 38 dígitos, aunque no se utilicen
CHAR	Tipo de dato de longitud fija. Se rellena con espacios en blanco por la derecha hasta completar la longitud especificada
VARCHAR	En desuso frente a VARCHAR2
VARCHAR2	Varchar2(100) reserva 100 bytes en memoria, aunque no se utilicen
INTEGER	Para números enteros. Si el valor va a estar comprendido entre $-2^{31} + 1$ y $2^{31} - 1$ se recomienda utilizar el subtipo PLS_INTEGER

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Instrucciones Técnicas	
	06.02. Normas de Construcción PL/SQL	

8.2. No abusar de las variables globales



Toda **variable** global definida en un paquete sólo se puede usar en el código de dicho paquete. Si desde otro objeto se quiere acceder a la misma, el paquete deberá proveer de una interfaz de manipulación. Esta variable estará declarada en el cuerpo del paquete y no en la especificación.

Los identificadores globales definidos en paquetes que sí podrán ser utilizados serán:

- *Constantes (siendo obligatorio especificar **CONSTANT**)*
- *Definición de tipos*

Es decir: Ninguna variable será definida en la especificación de un paquete. El acceso a la variable estará encapsulado por dos procedimientos uno que consulta el valor y otro que lo asigna.

Excepciones: Paquetes desempaquetados.



INCORRECTO

```
CREATE PACKAGE IH_PQ_XX IS
    INI NUMBER;
END;



PROCEDURE IH_PR_USA IS
BEGIN
    IH_PQ_XX.INI := IH_FU_HH();
    IF IH_PQ_XX.INI > 0 THEN ...
    ...
END;
```



CORRECTO

```
CREATE PACKAGE BODY IH_PQ_XX IS
    INI NUMBER;
    PR_ASIGNA( P NUMBER) IS ...
    FU_CONSULTA RETURN NUMBER IS ...
END;

PROCEDURE IH_PR_USA IS
BEGIN
    IH_PQ_XX.PR_ASIGNA(IH_FU_HH());
    IF IH_PQ_XX.FU_CONSULTA() > 0 THEN ...
    ...
END;
```

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Instrucciones Técnicas	
	06.02. Normas de Construcción PL/SQL	

8.3. Constantes SIRhUS.

8.3.1. Uso de constantes.



Debe hacerse uso de constantes para las tablas del tipo código/descripción. Para obtener el valor de una constante hay que utilizar la función `IH_FU_VALOR(cod_cte)`

Se definirá una constante para cualquier valor utilizado en el sistema de las tablas del tipo (código, descripción), también pueden utilizarse para los valores sobre dominios. No es necesario para las columnas de valores S/N.

8.3.2. Gestión de constantes.





Las constantes se almacenan en la tabla IHCONSTANTES.

Antes de solicitar el alta de una nueva constante hay que comprobar que no existe una constante para el valor y descripción.

- Las constantes serán gestionadas por una persona del equipo. A esta persona se le indicará el valor, la descripción y para que tabla/dominio se necesita. La persona dará de alta la constante y comunicará el código de la misma.





La gestión de las constantes está centralizada en una persona del equipo. A esta persona se le indicará el valor, la descripción y para que tabla/dominio se necesita. Esta persona comunicará el código asignado

	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	
	06. Instrucciones Técnicas	
	06.02. Normas de Construcción PL/SQL	

8.4. Eventos.

- Los eventos son procedimientos en b.d. con un formato (parámetros) y funcionalidad muy concreto, véase el documento eventos.doc
- El alta del evento en la tabla IHEVENTOS se deberá realizar mediante un script realizado por el desarrollador y que se adjuntará en la orden.
- La asociación del evento a la transición o acto se podrá realizar mediante script o dejar para el usuario.
- Los eventos devuelve el mensaje de error mediante un parámetro que puede ser numérico o cadena. En el primer caso el sistema se supondrá con que es un código APL-, en el segundo caso se deberá poner la cadena completa 'APL-99999...'

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Instrucciones Técnicas 06.02. Normas de Construcción PL/SQL	

9. Manipulando fechas

9.1. Comparación de Fechas.



En la comparación entre fechas hay que aplicar el calificador TO_DATE a aquellas cadenas o variables que no sean estrictamente de tipo fecha.

De no hacerlo, el resultado de la comparación dependerá de los formatos de fecha establecidos por defecto, tanto de Oracle como del entorno de ejecución.



EJEMPLO: Comparación entre fechas

```
SELECT ...
FROM ...
WHERE F_INICIO<=TO_DATE('05022004','DDMMYYYY')
```





EJEMPLO: comparación entre campos de tipos distintos

```
SELECT ...
FROM ...
WHERE F_INICIO <= '05/02/2004'
```

Caso de optar por la conversión en cadena se utilizará el formato año, mes y día.

```
TO_CHAR(F_1,'YYYYMMDD') <= TO_CHAR(F_2,'YYYYMMDD')
```

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Instrucciones Técnicas 06.02. Normas de Construcción PL/SQL	

9.2. Truncamiento de Fechas



Atención al parámetro empleado en la función **trunc**

El resultado de aplicar la función trunc a un campo tipo fecha dependerá del parámetro con el que lo utilizemos. Caso de no utilizar ninguno por defecto se trunca al día.



EJEMPLO: Cálculo del primer día del año

```
SELECT TRUNC(SYSDATE, 'YYYY')
FROM DUAL;
```

```
-- 01/01/2005 00:00:00
```



EJEMPLO: Cálculo del primer día del mes

```
SELECT TRUNC(SYSDATE, 'MM')
FROM DUAL;
```



```
-- 01/04/2005 00:00:00
```



EJEMPLO: Cálculo de la fecha y horas actuales

```
SELECT TRUNC (SYSDATE, 'HH')
FROM DUAL;
```

```
-- 18/04/2005 09:00:00
```

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Instrucciones Técnicas 06.02. Normas de Construcción PL/SQL	

9.3. Función LAST_DAY



La función LAST_DAY no implica el truncamiento de la hora





EJEMPLO: Último día del mes sin truncamiento

```
SELECT TO_CHAR(LAST_DAY(SYSDATE), 'DD/MM/YYYY HH24:MI')
FROM DUAL;
-----
30/04/2005 10:20
```



EJEMPLO: Último día del mes estricto

```
SELECT LAST_DAY(TRUNC(SYSDATE))
FROM DUAL;
-----
30/04/2005 00:00
```


 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Instrucciones Técnicas	
	06.02. Normas de Construcción PL/SQL	

9.4. Precaución en el uso del SYSDATE



En SIRhUS los campos de tipo fecha se almacenan en la base de datos sin referencia a las horas. Hay que contar con ello en el uso de SYSDATE.



Cualquier uso de SYSDATE sin truncar el día (horas y fracciones inferiores igual a cero), debe estar convenientemente justificado en los comentarios insertos en el código fuente.



Una excepción a esta norma son las fechas de auditoría y registros, que sí se almacenan en la base de datos de forma completa.





EJEMPLO: Sysdate. Uso incorrecto.

```
SELECT *
  FROM TABLA
 WHERE F_INIVIG <= SYSDATE
        AND F_FINIVIG >= SYSDATE
```



EJEMPLO: Sysdate. Uso correcto.

```
SELECT *
  FROM TABLA
 WHERE F_INIVIG <= TRUNC(SYSDATE)
        AND NVL(F_FINIVIG, TRUNC(SYSDATE)) >= TRUNC(SYSDATE)
```

	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	
	06. Instrucciones Técnicas 06.02. Normas de Construcción PL/SQL	

10. Control de Excepciones



Todo bloque de código en base de datos ha de tener un control de excepciones.

Toda ventana (*form*) ha de tener un *trigger on-error* a nivel de *form*, que realice la captura y tratamiento de excepciones.



Todo bloque de código asociado a una opción de menú ha de tener un control de excepciones, que debe incluir una llamada al procedimiento PR_MENU_EXCEPTION (biblioteca SIRLIB01).



Es recomendable distinguir entre excepciones de programa y excepciones funcionales



No se deben sobrescribir nombres de excepción predefinidos con excepciones definidas por el usuario.

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Instrucciones Técnicas	
	06.02. Normas de Construcción PL/SQL	

10.1. Excepciones en la sección declarativa



La sección de excepción de un bloque puede tratar únicamente los errores producidos en la sección ejecutable del mismo

Cuando se produce una excepción en la sección declarativa el control de ésta no puede ser tratada en el mismo bloque, sino que pasa al nivel superior. Por ello no es conveniente inicializaciones complejas de las variables.



EJEMPLO: Inicialización incorrecta.

```

DECLARE

    V_CODIGO NUMBER := OBTIENE_CODIGO;

BEGIN
.
.
.
EXCEPTION
.
END;
```



EJEMPLO: Inicialización en la sección ejecutable.



```

DECLARE

    V_CODIGO NUMBER := OBTIENE_CODIGO;

BEGIN
    V_CODIGO:= OBTIENE_CODIGO;

.
.
.
EXCEPTION
.
END;
```

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Instrucciones Técnicas	
		06.02. Normas de Construcción PL/SQL

10.2. Propagación de Excepciones





Es recomendable tratar una excepción dentro del bloque en el que se produce.

Las excepciones predefinidas las lanza el sistema, mientras que las excepciones definidas por el usuario se deben levantar manualmente (*RAISE*).

Cuando se levanta una excepción, si el bloque o subprograma actual no trata la excepción, se propaga al bloque superior. Este proceso se repite hasta que se encuentra un tratamiento dedicado a la excepción y en el caso de que ningún bloque trate la excepción *PL/SQL* devolverá un error del tipo *unhandled exception* al sistema. Las excepciones no se puede propagar a través de llamadas a procedimientos remotos.




Propagación de una excepción	
<pre> BEGIN BEGIN IF X = 1 THEN RAISE A; ELSIF X = 2 THEN RAISE B; ELSE RAISE C; END IF; ... EXCEPTION WHEN A THEN ... END; ... EXCEPTION WHEN B THEN ... END; </pre>	<ul style="list-style-type: none"> - La excepción A se trata en el bloque interno. - La excepción B se trata en el bloque externo. - La excepción C no se trata y provocará un "unhandled exception"


 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Instrucciones Técnicas	
	06.02. Normas de Construcción PL/SQL	

10.2.1. Uso de Raise_Application_Error con parámetros

El funcionamiento normal de la función Raise_Application_Error enmascara los distintos lanzamientos de excepciones, mostrando únicamente el último producido. Véase el siguiente ejemplo:



	EJEMPLO: Enmascaramiento de excepciones. <pre> 1 BEGIN 2 DECLARE 3 V NUMBER; 4 BEGIN 5 V := 1 / 0; 6 EXCEPTION 7 WHEN OTHERS THEN 8 RAISE_APPLICATION_ERROR(-20000, 'Error aquí'); 9 END; 10 EXCEPTION 11 WHEN OTHERS THEN 12 RAISE_APPLICATION_ERROR(-20000, 'Error general'); 13 END;</pre>
	<pre> ERROR en línea 1: ORA-20000: Error general ORA-06512: at line 12</pre>

Vemos como la segunda excepción oculta la anterior. Si queremos que este comportamiento no se produzca podemos utilizar un tercer parámetro en la función. Este parámetro nos permite controlar la lista de errores. Si el parámetro tiene como valor TRUE el error se agrega a la lista de errores producidos. En el caso de que el valor sea FALSE se produce el comportamiento de enmascaramiento normal de la función.

	EJEMPLO: Lista de excepciones. <pre> 1 BEGIN 2 DECLARE 3 V NUMBER; 4 BEGIN 5 V := 1 / 0; 6 EXCEPTION 7 WHEN OTHERS THEN 8 RAISE_APPLICATION_ERROR(-20000, 'Error aquí',true); 9 END; 10 EXCEPTION 11 WHEN OTHERS THEN 12 RAISE_APPLICATION_ERROR(-20000, 'Error general',true); 13 END;</pre>
	<pre> ERROR en línea 1: ORA-20000: Error general ORA-06512: at line 12 ORA-20000: Error aquí ORA-01476: divisor is equal to zero</pre>



Este comportamiento no altera la aparición de mensajes en la aplicación. Internamente sí aparece la pila completa de error.


	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	
	06. Instrucciones Técnicas 06.02. Normas de Construcción PL/SQL	

10.3. When Others



No es recomendable utilizar *When Others* para el tratamiento global de excepciones.



Para un tratamiento correcto de las excepciones producidas es recomendable tratar específicamente cada una de las excepciones posibles. El abuso de *When Others* puede enmascarar el error producido. Sólo se debe usar *When Others* sin acompañamiento de otras excepciones en el caso de que resulte indiferente que se produzca una excepción o cuando se requiera un tratamiento común de todas las excepciones posibles, o de un grupo de excepciones, sin poder o tener que especificar el nombre de cada excepción que se trata, dándole un tratamiento global a todas ellas.

Tratamiento de excepciones con WHEN OTHERS	
 <pre> BEGIN BEGIN IF X = 1 THEN RAISE A; ELSIF X = 2 THEN RAISE B; ELSE RAISE C; END IF; ... EXCEPTION WHEN A THEN ... END; ... EXCEPTION WHEN B THEN ... WHEN OTHERS THEN ... END;</pre>	<ul style="list-style-type: none"> - La excepción A se trata en el bloque interno. - La excepción B se trata en el bloque externo. - La excepción C se trata en la sección de tratamiento general de excepciones.



Es recomendable situar el tratamiento de *WHEN OTHERS* en el nivel máximo del programa PL/SQL

Hay que tener muy en cuenta el sistema de propagación de excepciones de *Oracle* para evitar que accidentalmente una excepción que deba tratarse a un nivel superior, quede tratada en la sección *WHEN OTHERS* de un nivel inferior. Preferiblemente los casos generales de excepciones con *WHEN OTHERS* deberían tratarse en los niveles superiores, y limitar su uso en niveles inferiores a casos específicos, en los que esté muy claro el objetivo que se persigue con el tratamiento general de todas las excepciones que no se hayan especificado.

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Instrucciones Técnicas	
	06.02. Normas de Construcción PL/SQL	





Tratamiento de excepciones con WHEN OTHERS	
<pre> BEGIN BEGIN IF X = 1 THEN RAISE A; ELSIF X = 2 THEN RAISE B; ELSE RAISE C; END IF; ... EXCEPTION WHEN A THEN ... WHEN OTHERS THEN ... END; ... EXCEPTION WHEN B THEN ... WHEN OTHERS THEN ... END;</pre>	<ul style="list-style-type: none"> - La excepción A se trata en el bloque interno. - Las excepciones B y C se tratan en la sección de tratamiento general de excepciones del nivel inferior. - NUNCA se podrá tratar la excepción B en el nivel superior dónde está su tratamiento específico



Tratamiento de excepciones con WHEN OTHERS	
<pre> BEGIN BEGIN IF X = 1 THEN RAISE A; ELSIF X = 2 THEN RAISE B; ELSE RAISE C; END IF; ... EXCEPTION WHEN A THEN ... WHEN C THEN ... END; ... EXCEPTION WHEN B THEN ... WHEN OTHERS THEN ... END;</pre>	<ul style="list-style-type: none"> - La excepción A se trata en el bloque interno. - La excepción B se trata en el bloque externo. - La excepción C se trata en el bloque interno. - La sección de tratamiento general de excepciones tratará otras excepciones no especificadas explícitamente.



En SIRhUS se hace uso del procedimiento IH_PR_OTHERS_EXCEPTION para gestionar las excepciones producidas.

 JUNTA DE ANDALUCÍA CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Instrucciones Técnicas	
	06.02. Normas de Construcción PL/SQL	

10.4. Pila de error y pila de llamadas



Para construir un sistema de localización de errores eficiente podemos utilizar el paquete **DBMS_UTILITY**. En concreto dos de sus funciones:

- **FORMAT_ERROR_STACK**
- **FORMAT_CALL_STACK**

La función **FORMAT_ERROR_STACK** nos puede ser útil para reproducir la pila de error. Con **DBMS_UTILITY.FORMAT_CALL_STACK** obtenemos la pila de llamadas, es decir, los distintos sitios desde los que se ha llamado a un determinado procedimiento o función.

EJEMPLO: Control de ejecución mediante DBMS_UTILITY.

```

>CREATE PROCEDURE B IS
  V NUMBER;
BEGIN
  DBMS_OUTPUT.PUT_LINE('*****');
  DBMS_OUTPUT.PUT_LINE('Estoy en B. he sido llamado por: ');
  DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_CALL_STACK);



  V := 1 / 0;
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('*****');
    DBMS_OUTPUT.PUT_LINE('Exception en B:');
    DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_ERROR_STACK);
    DBMS_OUTPUT.PUT_LINE('B: llamado por:');
    DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_CALL_STACK);
    RAISE_APPLICATION_ERROR(-20000,'Error en B',true);
END;

>CREATE PROCEDURE A IS
BEGIN
  B;
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('*****');
    DBMS_OUTPUT.PUT_LINE('Exception en A:');
    DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_ERROR_STACK);
END;

> EXEC A;

```



 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Instrucciones Técnicas	
	06.02. Normas de Construcción PL/SQL	

```

*****
Estoy en B. he sido llamado por:
----- PL/SQL Call Stack -----
   object      line  object
   handle      number name
38de7be30         6  procedure SADIEL22.B
38d674350         3  procedure SADIEL22.A
38b40c4a0         1  anonymous
block

*****
Exception en B:
ORA-01476: divisor is equal to zero

B: llamado por:
----- PL/SQL Call Stack -----
   object      line  object
   handle      number name
38de7be30        15  procedure SADIEL22.B
38d674350         3  procedure SADIEL22.A
38b40c4a0         1  anonymous
block



*****
Exception en A:
ORA-20000: Error en B
ORA-06512: at "SADIEL22.B", line 16
ORA-01476: divisor is equal to zero

Procedimiento PL/SQL terminado con éxito.

```



En el ejemplo anterior las llamadas de `raise_application_error` se realizan con el parámetro de mantenimiento de errores a `TRUE` para así obtener la pila de errores producidos

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Instrucciones Técnicas	
	06.02. Normas de Construcción PL/SQL	

10.5. Cerrar estructuras dinámicas de datos en Exception



En el tratamiento de Excepciones hay que prever el cierre de estructuras **dinámicas** de datos.

Cuando se haga uso de elementos creados dinámicamente se deben de prevenir el cierre en el bloque de excepciones que le corresponda, ya que el lenguaje PL/SQL no realiza una liberación y/o cierre implícito.

Los elementos a los que nos referimos son:



- Cursores definidos mediante el paquete DBMS_SQL.
- Fichero creado con el paquete UTL_FILE.

EJEMPLO CORRECTO

```

DECLARE
    CUR  PLS_INTEGER;
BEGIN
    CUR := DBMS_SQL.OPEN();
    <<aquí se produce una exception>>
    DBMS_SQL.CLOSE(CUR);
EXCEPTION
    -- hay que cerrar el cursor dinámico CUR
    IF DBMS_SQL.IS_OPEN(CUR) THEN
        DBMS_SQL.CLOSE(CUR);
    END IF;
END;
```



 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Instrucciones Técnicas	
	06.02. Normas de Construcción PL/SQL	

10.6. Gestión de errores en SIRhUS.





Hay una serie de consideraciones que hay que tener en cuenta para el tratamiento de los errores en SIRhUS

- Los errores, avisos, mensajes y demás de aplicación se deberán ir dando de alta en la medida que se necesiten. Estos mensajes de error se denominarán con el tipo APL, diferenciándolos de los FRM, OFG, ORA
- Antes de dar de alta un error se deberá verificar la no existencia del mismo. El mensaje se deberá tipificar según cuatro tipos :



TIPO	DESCRIPCION	PUSH BUTTON DEL ALERT
E	Mensaje de Error	Aceptar
D	Mensaje de Diálogo	Si, No
W	Warning o Aviso	Continuar, Cancelar
I	Mensaje de Información	Aceptar

- Al mismo tiempo que se anote el error en la orden se deberá dar de alta física con el mantenimiento de errores.
- En el mantenimiento se deberá introducir sólo el origen (Aplicación siempre) en código, la descripción y la descripción ampliada si procede, los CHECK no deben tocarse (los dos deben estar desmarcados). Deberá prestarse especial cuidado con la descripción corta ya que admite el carácter ` # ' siendo éste sustituido por los parámetros que procedan en su momento. Esta particularidad deberá usarse cuando se requiera que el mensaje tenga un contenido variable.
- El mantenimiento de la tabla de errores (IHERRORES) está centralizado en una persona, por lo que se deberá hablar con ella para crear/modificar/eliminar cualquiera de ellos.
- Es importante unificar el formato del texto informado en el mensaje así como su prosa. Se evitará a toda costa el uso de terminologías o tecnicismos no entendidos por un usuario (por ejemplo : violación de integridad referencial).
- Los mensajes se describirán con el grado de detalle preciso para dar la información a cualquier persona ajena al entorno informático. Si el error es funcional se deberá utilizar la terminología funcional adecuada.
- El texto deberá ser claro y conciso comenzando la frase en mayúsculas y continuándola en minúsculas completamente sin punto al final. Solo se iniciarán en mayúsculas aquellas palabras que para el sistema sean entidades concretas (Unidad, Persona, Puesto ...).
- Cuando se requiera un modo verbal proponiendo una acción se deberá hacer en modo presente (por ejemplo : Debe introducir un valor desde el 0 al 9).
- El código de excepción de usuario -20000 será el único que se usará para lanzar excepciones de usuario con mensajes de error APL.

	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	
	06. Instrucciones Técnicas	
	06.02. Normas de Construcción PL/SQL	

- La descripción del mensaje de error anterior tendrá el siguiente formato: 'APL-99999' o 'APL-99999#1datos#2datos#3datos', donde 99999 es el código de error, y en caso de tener parámetros, el texto a continuación de #1 es el primero, #2 el segundo, etc. Estos parámetros deberán pasarse de forma que la frase construida mantenga las reglas antes descritas.

En SIRhUS se disponen de unas herramientas para facilitar el control de errores producidos en la aplicación.

 JUNTA DE ANDALUCÍA CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Instrucciones Técnicas	
	06.02. Normas de Construcción PL/SQL	

10.6.1. **RAISE_APPLICATION_ERROR** en SIRhUS

El formato de esta instrucción es:

RAISE_APPLICATION_ERROR('código_error', 'texto_error', '¿mantenimiento_error?');



La llamada a esta función se debe realizar con los siguientes valores de los parámetros:

- **Código de error:** -20000 invariablemente
- **Texto de error:** Debe comenzar con el código de un mensaje de SIRHUS (APL-... sin espacios en blanco). En caso de requerir parámetros, éstos se pasarán mediante el uso del carácter de almohadilla y el número del parámetro. El número máximo de parámetros es de 4. No deben incluirse retornos de carro en el texto, en su lugar se utilizará el operador de concatenación (||).

Mantenimiento del error: Opcional. Ver [Definición de mensajes de error](#) para más detalles.



EJEMPLO: uso de Raise_Application_Error

```

BEGIN
  SELECT MIC_C_COLECTIVO,
         MIC_C_REGJUR,
         MIC_C_CATPER,
         MIC_C_MOTING
  INTO   P_C_COLECTIVO,
         P_C_REGJUR,
         P_C_CATPER,
         P_C_MOTING
  FROM   IHRELADM RAD
  WHERE  RAD.CAA_X_CABACTADM = P_X_CABACTADMRAD;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RAISE_APPLICATION_ERROR(-20000, 'APL-10297'
    || '#1 Relación con la Administración');
  WHEN OTHERS THEN
    -----
END;
```

← Falta un localizador para saber dónde está el error





EJEMPLO: uso de Raise_Application_Error

```

BEGIN
  SELECT MIC_C_COLECTIVO,
         MIC_C_REGJUR,
         MIC_C_CATPER,
         MIC_C_MOTING
  INTO   P_C_COLECTIVO,
         P_C_REGJUR,
         P_C_CATPER,
         P_C_MOTING
  FROM   IHRELADM RAD
  WHERE  RAD.CAA_X_CABACTADM = P_X_CABACTADMRAD;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RAISE_APPLICATION_ERROR (-20000, 'APL-10646 #1'
    || ' el motivo de ingreso.'
    || ' (IH_PQ_CHECH.IH_PR_MOTING)'
    || ' (IHRELADM)'
    || ' (' || TO_CHAR(P_X_CABACTADMRAD) || ')');
  WHEN OTHERS THEN
    -----
    -----
END;
```

← Localizador del error

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Instrucciones Técnicas	
	06.02. Normas de Construcción PL/SQL	

10.6.2.IH_PR_OTHERS_EXCEPTION



Este procedimiento analiza el contenido de la excepción producida. Si es un error provocado por el equipo de desarrollo lo propaga sin alterarlo, en caso contrario devuelve el error del tipo ORA producido.


Este procedimiento admite un único parámetro, en el que se recoge el bloque PL/SQL se ha provocado la excepción.



Ih_Pr_Others_Exception devuelve el localizador del error concatenado con el error producido.



EJEMPLO: Posible uso de IH_PR_OTHERS_EXCEPTION
<pre> CREATE OR REPLACE PROCEDURE IH_PR_PRUEBA IS BEGIN /* LO QUE SEA */ EXCEPTION WHEN NO_DATA_FOUND THEN RAISE_APPLICATION_ERROR(-20000, 'APL-XXXXX#1algo (IH_PR_PRUEBA)') ; WHEN OTHERS THEN IH_PR_OTHERS_EXCEPTION(' IH_PR_PRUEBA') ; END;</pre>

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Instrucciones Técnicas	
	06.02. Normas de Construcción PL/SQL	

EJEMPLO: Posible uso de IH_PR_OTHERS_EXCEPTION	
 <pre> /* IH_PR_DATOS_RAD */ ----- ----- BEGIN ----- ----- BEGIN SELECT MIC_C_COLECTIVO,MIC_C_REGJUR, MIC_C_CATPER,MIC_C_MOTING INTO V_C_COLECTIVO,V_C_REGJUR, V_C_CATPER, V_C_MOTING FROM IHRELADM RAD WHERE RAD.CAA_X_CABACTADM = P_X_CABACTADMRAD; EXCEPTION WHEN NO_DATA_FOUND THEN RAISE_APPLICATION_ERROR(-20000,'APL-10297' '#1 RELACIÓN CON LA ADMINISTRACIÓN'); WHEN OTHERS THEN IH_PR_OTHERS_EXCEPTION('IH_PR_DATOS_RAD- OBT.COL.REG. '); END; EXCEPTION WHEN OTHERS THEN /* Buena codificación/ IH_PR_OTHERS_EXCEPTION('IH_PR_DATOS_RAD'); /* Mala codificación/ RAISE_APPLICATION_ERROR(-20000,'APL-10297' '#1 RELACIÓN CON LA ADMINISTRACIÓN'); END IH_PR_DATOS_RAD; </pre>	<p>← Controlamos cualquier error que se produzca dentro de este bloque de datos .</p> <p>← Controlamos si se produce un error dentro del procedimiento y además conseguimos devolver el error original sin interferencias.</p> <p>← No nos a informar si el error se ha producido en algún procedimiento llamado desde éste.</p>

 JUNTA DE ANDALUCÍA CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Instrucciones Técnicas 06.02. Normas de Construcción PL/SQL	

10.6.3.IH_PR_MSG_OTHER_EXCP

Procedimiento en desuso. Tiene una funcionalidad limitada y una deficiente localización de errores. Devuelve la tabla afectada y el tipo de instrucción, más los 150 primeros caracteres del error.

Admite dos parámetros:

- **Tabla.** Nombre de la tabla que produce la excepción
- **Instrucción.** Tipo de instrucción que provoca la excepción. Los valores posibles son:
 - SEL. Selección o Consulta
 - INS. Inserción
 - MOD. Modificación o Actualización
 - ELI. Eliminación o Borrado





Caso de que el tipo de instrucción no sea uno de estos cuatro se devolverá el mensaje de código APL-10473.



EJEMPLO: Posible uso de IH_PR_MSG_OTHER_EXCP

```

BEGIN
  SELECT MIC_C_COLECTIVO, MIC_C_REGJUR,
         MIC_C_CATPER,    MIC_C_MOTING
  INTO   P_C_COLECTIVO,   P_C_REGJUR,
         P_C_CATPER,     P_C_MOTING
  FROM   IHRELADM RAD
  WHERE  RAD.CAA_X_CABACTADM = P_X_CABACTADM;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    IH_PR_MSG_OTHERS_EXCP('IHRELADM', 'SEL');
END;
```


 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Instrucciones Técnicas	
	06.02. Normas de Construcción PL/SQL	

10.6.4.IH_PR_TEXTO_ERROR

Este procedimiento tiene como parámetro una cadena de texto. Devuelve el código de error y el texto del error formateado adecuadamente. Útil dentro de forms y reports.





```

EJEMPLO: Uso de IH_PR_TEXTO_ERROR dentro de un informe

BEGIN
  ---
  EXCEPTION
  WHEN OTHERS THEN
    DECLARE
      CODIGO NUMBER;
      MENSAJE VARCHAR2(2000);
    BEGIN
      IH_PR_TEXTO_ERROR(SQLERRM,CODIGO,MENSAJE);
      SRW.MESSAGE(CODIGO,MENSAJE);
    END;
    ---
  END;

```

	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	
	06. Instrucciones Técnicas	
	06.02. Normas de Construcción PL/SQL	



10.6.5. Normalización de los códigos de error.

Existen unos rangos de códigos de error en función del subsistema afectado:

Rango	Subsistema
00000 - 09999	Subsistema X y Genéricos
10000 - 19999	Subsistema A
20000 - 29999	Subsistema B
30000 - 39999	Subsistema C
40000 - 49999	Subsistema D
60000 - 69999	Subsistema F
70000 - 79999	SIRHUS Canarias
80000 - 89999	Web del Empleado
90000 - 99999	Subsistema G



No deben utilizarse mensajes de error en un subsistema definidos en otro subsistema.

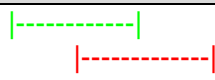
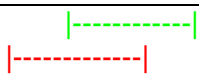



 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Instrucciones Técnicas	
	06.02. Normas de Construcción PL/SQL	

11. Consultas Prácticas



11.1. Solapamiento de fechas.

Vamos a analizar cómo comprobar si un rango de fechas determinado se solapa con los ya existentes.



Existen 5 posibilidades diferentes de solapes de fechas.

Caso	Representación
1 Fecha de Inicio dentro de un Rango	<div> <div>fecha existente</div> <div>nuevo rango</div> </div> 
2 Fecha Final dentro de un Rango	<div> <div>fecha existente</div> <div>nuevo rango</div> </div> 
3 Fecha de inicio y Final dentro de dos Rangos diferentes	<div> <div>fecha existente</div> <div>nuevo rango</div> <div>fecha existente</div> </div> 
4 Fecha de inicio y Final dentro de un mismo rango	<div> <div>fecha existente</div> <div>nuevo rango</div> </div> 
5 Rango incluido en un rango existente	<div> <div>fecha existente</div> <div>nuevo rango</div> </div> 

Esta casuística se complica puesto que fecha final normalmente puede ser nula .

 JUNTA DE ANDALUCÍA CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Instrucciones Técnicas	
	06.02. Normas de Construcción PL/SQL	

Solapamiento de Fechas. Solución de todos los casos posibles
<pre> /***** /* queremos comprobar que para el rango */ /* de fechas f_inicio-v_f_fin no */ /* existan solapes de fecha dentro de */ /* nuestra tabla para un código */ *****/ DECLARE CURSOR CU_COMFECH(P_CODIGO TABLA.C_CODIGO%TYPE, P_F_INICIO DATE, P_F_FIN DATE) IS SELECT 1 FROM TABLA WHERE C_CODIGO = P_CODIGO AND F_INICIO <= NVL(P_F_FIN, F_INICIO) AND NVL(F_FIN, P_F_INICIO) >= P_F_INICIO; V_CODIGO NUMBER; V_ERROR NUMBER; V_F_INICIO DATE; V_F_FIN DATE; BEGIN /*comprobar si hay solape */ V_ERROR := 0; OPEN CU_COMFECH(V_CODIGO,V_F_INICIO,V_F_FIN); FETCH CU_COMFECH INTO V_ERROR; IF CU_COMFECH%FOUND THEN ... -- existe solape de fecha ELSE ... -- no hay solape de fechas END IF; CLOSE CU_COMFECH; END;</pre>

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Instrucciones Técnicas	
	06.02. Normas de Construcción PL/SQL	

11.2. Cálculo de los años, meses y días transcurridos entre dos fechas.



Al realizar diferencias entre fechas Oracle devuelve el número de días transcurridos.

La función MONTHS_BETWEEN devuelve el número de meses, con decimales, transcurridos entre dos fechas.



```

Años, meses y días que transcurren entre P_FECHA1 y P_FECHA2
PROCEDURE DMA_BETWEEN
( P_FECHA1 DATE, P_FECHA2 DATE,
  P_ANIOS OUT NUMBER, P_MESES OUT NUMBER, P_DIAS OUT NUMBER) IS

  V_N_MESES NUMBER;

BEGIN

  V_N_MESES := MONTHS_BETWEEN(P_FECHA1, P_FECHA2);

  P_ANIOS := FLOOR( V_N_MESES / 12);
  P_MESES := MOD( FLOOR(V_N_MESES), 12);
  P_DIAS := ROUND( (V_N_MESES - FLOOR(V_N_MESES))*31);



END;
```



Es recomendable que p_fecha2 sea mayor que p_fecha1



En el ejemplo anterior p_fecha1 está incluido en el intervalo, no así p_fecha2. Para incluirlo habría que reemplazarlo por p_fecha2 + 1

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Instrucciones Técnicas 06.02. Normas de Construcción PL/SQL	

11.3. Cálculo de los días laborables incluidos entre dos fechas.

La siguiente función implementa este cálculo. En este procedimiento se han considerado como días no laborables los Sábados, Domingos y festivos de Lunes a Viernes.



```

EJEMPLO: Años, meses y días que transcurren entre P_FECHA1 y P_FECHA2
FUNCTION FU_DIFFLAB (P_F_DESDE DATE, P_F_HASTA DATE)
RETURN NUMBER IS

    V_N_NATURALES    NUMBER;
    V_N_DIAS_FEST     NUMBER;
    V_N_DIAS_LAB      NUMBER;
    V_F_LUNES_DESDE   DATE;

BEGIN
-- Ojo: la p_f_desde la movemos hasta el lunes de esa semana con: TRUNC(, 'w')
-- El motivo de mover la fecha desde al lunes es para poner dividir por 7
-- y contar esa semana como 1.
    V_F_LUNES_DESDE := TRUNC (P_F_DESDE, 'W');

-- número de días naturales entre el lunes_desde y p_f_hasta inclusive.
    V_N_NATURALES := P_F_HASTA - V_F_LUNES_DESDE + 1;

-- Número de sábados y domingos entre el lunes desde y fecha hasta.
    V_N_DIAS_FEST := FLOOR (V_N_NATURALES / 7) * 2;

-- si la fecha desde es domingo descontamos un día por el sábado de esa semana
-- contado
    IF (P_F_DESDE - V_F_LUNES_DESDE + 1) = 7 /* FU_ES_DOMINGO (P_F_DESDE) */ THEN
        V_N_DIAS_FEST := V_N_DIAS_FEST - 1;
    END IF;

-- si la fecha hasta es sábado hay que sumar uno porque no se ha tenido en cuenta.
    IF (P_F_HASTA - TRUNC(P_F_HASTA, 'W') + 1) = 6 /* FU_ES_SABADO(P_F_HASTA) */ THEN
        V_N_DIAS_FEST := V_N_DIAS_FEST + 1;
    END IF;

-- obtener el número de días festivos (según calendario laboral)
-- que caen de lunes a viernes y sumar.
    V_N_DIAS_FEST := V_N_DIAS_FEST + FU_FESTIVOS_EN_LAB (P_F_DESDE, P_F_HASTA);

-- el número de días laborables es el número de días naturales menos los festivos.
    V_N_DIAS_LAB := (P_F_HASTA - P_F_DESDE + 1) - V_N_DIAS_FEST;

    RETURN V_N_DIAS_LAB;

END;
```



Este algoritmo hace referencia a la función `fu_festivos_en_lab (p_f_desde, p_f_hasta)`, que informa de los días festivos según el calendario laboral.