5. Manejo de Errores

Este tema se centra en el tratamiento de las excepciones. En PL/SQL una advertencia o condición de error es llamada una *excepción*. Estas pueden ser definidas en forma interna (en tiempo de ejecución de un programa) o explícitamente por el usuario.

- Cuando ocurre un error se alcanza la excepción, esto quiere decir que se ejecuta la porción del programa donde ésta se encuentra implementada, transfiriéndose el control a ese bloque de sentencias.
- Las excepciones definidas por el usuario deben ser alcanzadas explícitamente utilizando la sentencia *raise*.
- Con las excepciones ser pueden manejar los errores cómodamente sin necesidad de mantener múltiples chequeos por cada sentencia escrita.
- También provee claridad en el código desde el momento en que permite mantener las rutinas correspondientes al tratamiento de los errores en forma separada de la lógica del negocio.

```
DECLARE
 -- Declaraciones
BEGIN
  -- Ejecución
EXCEPTION
  -- Excepción
END;
```

- Cuando ocurre un error, se ejecuta la porción del programa marcada por el bloque **EXCEPTION**, transfiriéndose el control a ese bloque de sentencias.

Excepciones predefinidas

- Las excepciones predefinidas no necesitan ser declaradas. Simplemente se utilizan cuando estas son gatilladas por algún error determinado.
- La siguiente es la lista de las excepciones predeterminadas por PL/SQL y una breve descripción de cuándo son accionadas:

Nombre Excepción	Gatillada cuando…	SQLCODE
ACCESS_INTO_NULL	El programa intentó asignar valores a los atributos de un objeto no inicializado	-6530
COLLECTION_IS_NULL	El programa intentó asignar valores a una tabla anidada aún no inicializada	-6531
CURSOR_ALREADY_OPEN	El programa intentó abrir un cursor que ya se encontraba abierto. Recuerde que un cursor de ciclo FOR automáticamente lo abre y ello no se debe especificar con	-6511

	la sentencia OPEN	
DUP_VAL_ON_INDEX	El programa intentó almacenar valores duplicados en una columna que se mantiene con restricción de integridad de un índice único (unique index)	-1
INVALID_CURSOR	El programa intentó efectuar una operación no válida sobre un cursor	-1001
INVALID_NUMBER	En una sentencia SQL, la conversión de una cadena de caracteres hacia un número falla cuando esa cadena no representa un número válido	-1722
LOGIN_DENIED	El programa intentó conectarse a Oracle con un nombre de usuario o password inválido	-1017
NO_DATA_FOUND	Una sentencia SELECT INTO no devolvió valores o el programa referenció un elemento no inicializado en una tabla indexada	+100
NOT_LOGGED_ON	El programa efectuó una llamada a Oracle sin estar conectado	-1012
PROGRAM_ERROR	PL/SQL tiene un problema interno	-6501
ROWTYPE_MISMATCH	Los elementos de una asignación (el valor a asignar y la variable que lo contendrá) tienen tipos incompatibles. También se presenta este error cuando un parámetro pasado a un subprograma no es del tipo esperado	-6504
SELF_IS_NULL	El parámetro SELF (el primero que es pasado a un método MEMBER) es nulo	-30625
STORAGE_ERROR	La memoria se terminó o está corrupta	-6500
SUBSCRIPT_BEYOND_COUNT	El programa está tratando de referenciar un elemento de un arreglo indexado que se encuentra en una posición más grande que el número real de elementos de la colección	-6533
SUBSCRIPT_OUTSIDE_LIMIT	El programa está referenciando un elemento de un arreglo utilizando un número fuera del rango permitido (por ejemplo, el elemento “-1”)	-6532
SYS_INVALID_ROWID	La conversión de una cadena de caracteres hacia un tipo *rowid* falló porque la cadena no representa un número	-1410
TIMEOUT_ON_RESOURCE	Se excedió el tiempo máximo de espera por un recurso en Oracle	-51
TOO_MANY_ROWS	Una sentencia SELECT INTO devuelve más de una fila	-1422
VALUE_ERROR	Ocurrió un error aritmético, de conversión o truncamiento. Por ejemplo,	-6502

	sucede cuando se intenta calzar un valor muy grande dentro de una variable más pequeña	
ZERO_DIVIDE	El programa intentó efectuar una división por cero	-1476

•El siguiente ejemplo muestra un bloque de excepciones que captura las excepciones **NO_DATA_FOUND** y **ZERO_DIVIDE.** Cualquier otra excepción será capturada en el bloque **WHEN OTHERS THEN**.

```
DECLARE
 -- Declaraciones
BEGIN
  -- Ejecución
EXCEPTION
WHEN NO_DATA_FOUND THEN
  -- Se ejecuta cuando ocurre una excepción de tipo NO_DATA_FOUND
WHEN ZERO_DIVIDE THEN
  -- Se ejecuta cuando ocurre una excepción de tipo ZERO_DIVIDE
WHEN OTHERS THEN
  -- Se ejecuta cuando ocurre una excepción de un tipo no tratado
-- en los bloques anteriores
END;
```

•Como ya hemos dicho cuando ocurre un error, se ejecuta el bloque **EXCEPTION**, transfiriéndose el control a las sentencias del bloque.
•Una vez finalizada la ejecución del bloque de **EXCEPTION** no se continúa ejecutando el bloque anterior.
• Si existe un bloque de excepción apropiado para el tipo de excepción se ejecuta dicho bloque.
•Si no existe un bloque de control de excepciones adecuado al tipo de excepción se ejecutará el bloque de excepción **WHEN OTHERS THEN** (si existe!).
•**WHEN OTHERS** debe ser el último manejador de excepciones.

Excepciones definidas por el usuario

•PL/SQL permite al usuario definir sus propias excepciones, las que deberán ser declaradas y gatilladas explícitamente utilizando otros comandos del lenguaje.
•Las excepciones sólo pueden ser declaradas en el segmento "Declare" de un bloque, subprograma o paquete.
•Se declara una excepción escribiendo su nombre seguida de la palabra clave EXCEPTION.
•Las declaraciones son similares a las de variables, pero recuerde que una excepción es una condición de error, no un ítem de datos.
•Aun así, las mismas reglas de alcance aplican tanto sobre variables como sobre las excepciones.

•Ejemplo:

```
DECLARE
  error_01      EXCEPTION;
```

Reglas de Alcance

•Una excepción no puede ser declarada dos veces en un mismo bloque.

•Tal como las variables, una excepción declarada en un bloque es local a ese bloque y global a todos los sub-bloques que comprende.

La sentencia "RAISE"

•La sentencia RAISE permite gatillar una excepción en forma explícita.

•Es factible utilizar esta sentencia en cualquier lugar que se encuentre dentro del alcance de la excepción.

•Ejemplos:

```
DECLARE
  out_of_stock        EXCEPTION;          -- declaración de la excepción
  total           NUMBER(4);
BEGIN
  …
  IF total < 1 THEN
        RAISE out_of_stock;               -- llamado a la excepción
  END IF;
EXCEPTION
  WHEN out_of_stock THEN
        -- manejar el error aquí
  WHEN OTHERS THEN
        …
END;
```

```
DECLARE
 -- Declaramos una excepción identificada por VALOR_NEGATIVO
VALOR_NEGATIVO        EXCEPTION;
valor                 NUMBER;
BEGIN
  -- Ejecución
valor := -1;
IF valor < 0 THEN
RAISE VALOR_NEGATIVO;
END IF;
EXCEPTION
  -- Excepción
```

```
WHEN VALOR_NEGATIVO THEN
dbms_output.put_line('El valor no puede ser negativo');
END;
```

Excepción OTHERS

•Finalmente, cabe destacar la existencia de la excepción OTHERS, que simboliza cualquier condición de excepción que no ha sido declarada.

•Se utiliza comúnmente al final del bloque de excepciones para absorber cualquier tipo de error que no ha sido previsto por el programador.

•En ese caso, es común observar la sentencia ROLLBACK en el grupo de sentencias de la excepción o alguna de las funciones:

- •SQLCODE
- •SQLERRM

Uso de SQLCODE y SQLERRM

•Al manejar una excepción es posible apoyarse con las funciones predefinidas SQLCode y SQLErrm para aclarar al usuario la situación de error acontecida.

•*Sqlcode* siempre retornará el número del error de Oracle y un "0" (cero) en caso exitoso al ejecutarse una sentencia SQL.

•Por otra parte, Sqlerrm retornará el correspondiente mensaje de error para la situación ocurrida.

•Estas funciones son muy útiles cuando se utilizan en el bloque de excepciones, para aclarar el significado de la excepción OTHERS, cuando ésta ocurre.

•Estas funciones no pueden ser utilizadas directamente en una sentencia SQL

•Se puede asignar su valor a alguna variable de programa y luego usar esta última en alguna sentencia.

•Ejemplo:

```
DECLARE
  err_num      NUMBER;
  err_msg      VARCHAR2(100);

BEGIN
  …
EXCEPTION
  WHEN OTHERS THEN
        err_num := SQLCODE;
        err_msg := SUBSTR(SQLERRM, 1, 100);
        INSERT INTO errores VALUES(err_num, err_msg);
END;
```

- Con SQL*PLUS:

```
SQL> DECLARE
  2  err_num NUMBER;
  3  err_msg VARCHAR2(255);
  4  result  NUMBER;
  5  BEGIN
  6    SELECT 1/0 INTO result   FROM DUAL;
  7  EXCEPTION
  8  WHEN OTHERS THEN
  9  err_num := SQLCODE;
 10  err_msg := SQLERRM;
 11  DBMS_OUTPUT.put_line('Error:'||TO_CHAR(err_num));
 12  DBMS_OUTPUT.put_line(err_msg);
 13  END;
 14  /
Error:-1476
ORA-01476: el divisor es igual a cero
Procedimiento PL/SQL terminado con éxito.
```

•También es posible entregarle a la función SQLERRM un número negativo que represente un error de Oracle y ésta devolverá el mensaje asociado.

RAISE_APPLICATION_ERROR

• En ocasiones queremos enviar un mensaje de error personalizado al producirse una excepción PL/SQL.

•Para ello es necesario utilizar la instrucción **RAISE_APPLICATION_ERROR**;

•La sintaxis general es la siguiente:

RAISE_APPLICATION_ERROR(<error_num>,<mensaje>);

Siendo:

•error_num un entero negativo comprendido entre -20000 y -20999

•Mensaje es la descripción del error con una longitud no superior a 2048 caracteres.

```
DECLARE
  v_div NUMBER;
BEGIN
    SELECT 1/0 INTO v_div FROM DUAL;
EXCEPTION
  WHEN OTHERS THEN
  RAISE_APPLICATION_ERROR(-20001,'No se puede dividir por cero');
END;
```

•En SQL*PLUS:

```
SQL> DECLARE
  2  v_div NUMBER;
  3  BEGIN
  4  SELECT 1/0 INTO v_div FROM DUAL;
  5  EXCEPTION
  6  WHEN OTHERS THEN
  7  RAISE_APPLICATION_ERROR(-20001,'No se puede dividir por cero');
  8  END;
  9  /
DECLARE
*
ERROR en línea 1:
ORA-20001: No se puede dividir por cero
ORA-06512: en  línea 7
```

–SIEMPRE APARECE CON RAISE_APPLICATION_ERROR EL ORA-06512

•Si RAISE_APPLICATION_ERROR es llamado por un bloque PL/SQL, la ejecución de dicho bloque finaliza y todas las modificaciones realizadas sobre la BD son anuladas.

•Dicho de otro modo, se desarrolla un ROLLBACK implícito además de mostrar el mensaje de error.

Propagación de excepciones en PL/SQL

•Una de las características más interesantes de las excepciones es la propagación de excepciones.

•Cuando se lanza una excepción, el control se transfiere hasta la sección **EXCEPTION** del bloque donde se ha producido la excepción.

•Entonces se busca un manejador válido de la excepción (**WHEN** <*excepción*> **THEN**, **WHEN OTHERS THEN**) dentro del bloque actual.

•En el caso de que no se encuentre ningún manejador válido el control del programa se desplaza hasta el bloque EXCEPTION del bloque que ha realizado la llamada PL/SQL.

•Observemos el siguiente bloque de PL/SQL (Nótese que se ha añadido una cláusula WHERE 1=2 para provocar una excepción **NO_DATA_FOUND**).

EJEMPLO EN SQL*PLUS:

```
SQL> CREATE OR REPLACE FUNCTION fn_fecha RETURN DATE  IS fecha DATE;
  2  BEGIN
  3  SELECT SYSDATE INTO fecha FROM DUAL
  4  WHERE 1=2;
  5  RETURN fecha;
  6  EXCEPTION
  7  WHEN ZERO_DIVIDE THEN
  8  dbms_output.put_line('EXCEPCION ZERO_DIVIDE CAPTURADA EN fn_fecha');
  9  END;
 10  /
```

Función creada.

```
SQL> DECLARE
  2  fecha DATE;
  3  BEGIN
  4     fecha := fn_fecha;
  5  dbms_output.put_line('La fecha es '||TO_CHAR(fecha, 'DD/MM/YYYY'));
  6  EXCEPTION
  7  WHEN NO_DATA_FOUND THEN
  8  dbms_output.put_line('EXCEPCIÓN NO_DATA_FOUND CAPTURADA EN  EL
BLOQUE PRINCIPAL');
  9  END;
 10  /
EXCEPCIÓN NO_DATA_FOUND CAPTURADA EN  EL BLOQUE PRINCIPAL
```

Procedimiento PL/SQL terminado con éxito.

•La excepción **NO_DATA_FOUND** se produce durante la ejecución de la función fn_fecha, pero como no existe ningún manejador de la excepción en dicha función la excepción se propaga hasta el bloque que ha realizado la llamada. En ese momento se captura la excepción.