UNIDAD DIDÁCTICA 6

PROGRAMACIÓN DE BASES DE DATOS

MÓDULO PROFESIONAL: BASES DE DATOS

Índice

RESUMEN INTRODUCTORIO 2
INTRODUCCIÓN 2
CASO INTRODUCTORIO 3
1. ORACLE Y PL/SQL 4
2. CARACTERÍSTICAS GENERALES 8
3. TIPOS DE DATOS 9
4. OPERADORES 10
5. ESTRUCTURAS DE CONTROL 11
5.1 Estructuras de control de flujo 11
5.2 Estructuras repetitivas 12
6. PROCEDIMIENTOS ALMACENADOS 15
6.1 Sintaxis 15
6.2 Parámetros 15
6.3 Estructura de un Bloque 17
6.4 Borrar un procedimiento almacenado 17
6.5 Declaración de Variables 18
7. FUNCIONES Y FUNCIONES DEFINIDAS POR EL USUARIO 20
8. EVENTOS: DISPARADORES O TRIGGERS 23
9. CURSORES 27
10. EXCEPCIONES 31
11. AUTOMATIZACIÓN DE TAREAS 37
RESUMEN FINAL 40

RESUMEN INTRODUCTORIO

En la presente unidad se estudiarán conceptos fuera del ámbito relacional de las bases de datos, para introducir la programación en ellas. En este sentido, se analizan los tipos de datos utilizables para la implementación de funciones, los diferentes operadores que se pueden aplicar, las estructuras de control que, al igual que en cualquier otro lenguaje de programación, serán del tipo control de flujo o/y repetitivas. Además, se detallan los procedimientos almacenados, su funcionalidad, la sintaxis correcta, qué parámetros y qué estructura presenta, así como algunas acciones de interés como la creación, la declaración de variables y el borrado de dichos procedimientos. Por último, se tratarán otros contenidos como los cursores, las excepciones (tanto predefinidas como definidas por el usuario) y los disparadores o triggers.

INTRODUCCIÓN

La gestión eficiente de bases de datos es esencial para el funcionamiento y rendimiento óptimo de sistemas empresariales y aplicaciones de software. En este contexto, Oracle y PL/SQL desempeñan un papel crucial al ofrecer un robusto sistema de gestión de bases de datos y un lenguaje de programación procedural específico para la base de datos Oracle. Este conjunto de herramientas proporciona a los desarrolladores y administradores de bases de datos las capacidades necesarias para diseñar, implementar y mantener sistemas de bases de datos escalables y altamente fiables.

Por ello, conocer desde los aspectos generales hasta detalles algo más específicos de programación y administración, como las características generales de Oracle, la automatización de tareas y la gestión de excepciones, es fundamental. También, comprender la gran importancia de conceptos como los tipos de datos, operadores, procedimientos almacenados, funciones y demás elementos capacitará a los profesionales para desarrollar aplicaciones eficientes y mantener la integridad de los datos en entornos empresariales críticos.

CASO INTRODUCTORIO

La consultoría para la que trabaja tu amiga Paula requiere actualizar su base de datos e implementar una serie de procedimientos que le permitan trabajar con los datos que tiene almacenados. Necesitan distinguir entre los clientes del sector empresarial privado y los clientes de la administración pública. Para automatizar esta acción, Paula te pide que te encargues de crear un procedimiento almacenado implementando la sintaxis correcta. De esta manera, cada vez que se firme un contrato con un nuevo cliente, sus datos serán almacenados de forma automática en una tabla u otra, dependiendo del tipo de cliente. Sin un procedimiento almacenado sería una ardua tarea para el departamento de administración el tener que ir cambiando de tabla cada vez que se llega a un acuerdo con un nuevo cliente, teniendo presente en todo momento la naturaleza del mismo.

Al finalizar la unidad el alumnado conocerá la sintaxis del lenguaje de programación PL-SQL, y será capaz de realizar programas que manipulen datos dentro de una base de datos, sabiendo cómo sacar el máximo partido a las mismas mediante el uso de la programación.

1. ORACLE Y PL/SQL

Los jefes de tu empresa han decidido volver a utilizar una aplicación que tenían aparcada desde hace mucho tiempo, ya que creen que les puede resultar útil. La base de datos de dicha aplicación, que es una base de datos Oracle, cuenta con una serie de procedimientos almacenados y funciones. Paula te pide que te pongas al día con el lenguaje PL/SQL y revises el código de dichos procedimientos y funciones.

SQL es un lenguaje de consulta para los sistemas de bases de datos relacionales, pero que no posee la potencia de los lenguajes de programación. No permite el uso de variables, estructuras de control de flujo, bucles, y demás elementos característicos de la programación. No es de extrañar, SQL es un lenguaje de consulta, no un lenguaje de programación.

Los procedimientos almacenados amplían SQL con los elementos característicos de los lenguajes de programación, variables, sentencias de control de flujo, bucles, etc. Cuando se desea realizar una aplicación completa para el manejo de una base de datos relacional, resulta necesario utilizar alguna herramienta que soporte la capacidad de consulta del SQL y la versatilidad de los lenguajes de programación tradicionales. **PL/SQL** es el lenguaje de programación que proporciona **Oracle** para extender el SQL estándar con otro tipo de instrucciones y elementos propios de los lenguajes de programación.

Cabe mencionar que Oracle es un Sistema Gestor de Bases de Datos Objeto-Relacional (SGBDOR) que incorpora conceptos del paradigma orientado a objetos. Por tanto, un Sistema de Gestión Objeto-Relacional contiene ambas tecnologías: relacional y de objetos.

PARA SABER MÁS

Amplía la información sobre este lenguaje de programación:

VÍDEO DE INTERÉS

Visualiza cómo instalar Oracle Database 11G Express Edition:

Ahora conoceremos las herramientas para trabajar con Oracle:

En el vasto panorama de la gestión de bases de datos, Oracle se destaca como una de las opciones más robustas y ampliamente utilizadas. Para optimizar la experiencia de desarrollo y administración, es imperativo familiarizarse con herramientas especializadas que faciliten estas tareas. A continuación, se enumeran tres de las principales herramientas de escritorio para trabajar con Oracle:

Oracle SQL Developer

Oracle SQL Developer es una herramienta de desarrollo y administración de bases de datos diseñada para facilitar las tareas relacionadas con bases de datos Oracle. Desarrollada por Oracle Corporation, esta aplicación proporciona un entorno gráfico integral que permite a los desarrolladores, DBAs (Administradores de Bases de Datos) y analistas interactuar de manera efectiva con las bases de datos Oracle. Su interfaz intuitiva no solo agiliza el desarrollo, sino que también ofrece características avanzadas para la gestión eficiente de bases de datos Oracle.

ENLACE DE INTERÉS

Aquí podrás descargar SQL Developer:

PL/SQL Developer

Adentrándonos en el universo de la programación en Oracle, nos encontramos con PL/SQL Developer. Esta herramienta se erige como una plataforma dedicada para la creación y depuración de procedimientos almacenados, funciones y desencadenadores en Oracle. Con funcionalidades específicas para el desarrollo eficaz de código PL/SQL, PL/SQL Developer se convierte en una herramienta muy interesante para aquellos que buscan optimizar y perfeccionar sus scripts en la base de datos Oracle.

ENLACE DE INTERÉS

Aquí podrás descargar PL/SQL Developer:

DBeaver

Es una herramienta de administración de bases de datos de código abierto que proporciona soporte para Oracle, entre otros sistemas de gestión de bases de datos. DBeaver se destaca por su versatilidad al admitir múltiples sistemas de bases de datos en una sola interfaz, permitiendo a los usuarios gestionar, explorar y consultar bases de datos Oracle de manera eficiente. Su naturaleza multiplataforma y su capacidad para adaptarse a diversas tecnologías hacen de DBeaver una opción atractiva para aquellos que trabajan en entornos heterogéneos.

ENLACE DE INTERÉS

Aquí podrás descargar DBeaver:

2. CARACTERÍSTICAS GENERALES

Paula te dice que, para seguir con tu formación en PL/SQL, hay una serie de conceptos o características generales que debes conocer. Para ello, te dispones a investigar sobre las distintas unidades léxicas y su clasificación.

De forma resumida, se analizan algunas características de los procedimientos almacenados. Una línea contiene grupos de caracteres conocidos como *unidades léxicas*, que pueden ser clasificadas como:

- **Delimitador**: se refiere a un carácter o conjunto de caracteres que se utiliza para separar las diferentes instrucciones o comandos en una secuencia de comandos SQL, que tiene una función especial en los procedimientos almacenados. Estos pueden ser: Operadores Aritméticos, Operadores Lógicos y Operadores Relacionales.

- **Identificador**: se refiere a un nombre que se utiliza para identificar un elemento dentro de la base de datos, como tablas, columnas, índices, procedimientos almacenados, y otros objetos. Los identificadores son esenciales para referirse y manipular datos de manera efectiva en una base de datos. Estas unidades y objetos incluyen: Constantes, cursores, variables, subprogramas y excepciones.

- **Literal**: es un valor de tipo numérico, carácter, cadena o lógico no representado por un identificador (es un valor explícito).
- **Comentario**: es una aclaración que el programador incluye en el código. Son soportados 2 estilos de comentarios, de línea simple y de multilínea, para lo cual son empleados ciertos caracteres especiales:

```
-- línea simple
/* Conjunto de Líneas
*/
```

3. TIPOS DE DATOS

Nuevamente, te han asignado un ayudante para realizar más rápido el trabajo. Como tiene poca experiencia, procedes a explicarle los tipos de datos que hay en Oracle y cuáles son sus formatos.

Cada constante y variable tiene un tipo de dato en el cual se especifica el formato de almacenamiento, restricciones y rango de valores válidos. Casi todos los tipos de datos manejados por los procedimientos almacenados son similares a los soportados por SQL. A continuación, se muestran los más comunes:

- **NUMERIC** (Numérico): número en coma flotante desempaquetado. El número se almacena como una cadena: saldo NUMERIC(16,2).

- **CHAR** (Carácter): este tipo se utiliza para almacenar cadenas de longitud fija. Su longitud abarca desde 1 a 255 caracteres: nombre CHAR(20).

- **VARCHAR** (Carácter de longitud variable): al igual que el anterior, se utiliza para almacenar cadenas, en el mismo rango de 1-255 caracteres, pero en este caso, de longitud variable: dirección VARCHAR(50).

- **DATE** (Fecha): datos de tipo fecha. El formato por defecto es YYYY MM DD.

- **DATETIME** (Combinación de fecha y hora): el formato de almacenamiento es de año-mes-día horas:minutos:segundos.

PARA SABER MÁS

Visualiza los diferentes tipos de datos en PL-SQL:

ENLACE DE INTERÉS

Es muy recomendable consultar los aspectos sobre el ámbito y visibilidad de las variables:

4. OPERADORES

Para completar la formación de tu ayudante, le pides que se ponga al día con los operadores que se usan en Oracle ya que, aunque son muy parecidos a los de MySQL, existen algunas diferencias.

La siguiente tabla ilustra los operadores disponibles.

Tipo de Operador	Operador	
Asignación	:=	Asignación
Aritméticos	+	Suma
	-	Resta
	*	Multiplicación
	/	División
	**	Exponente
Relacionales o de Comparación	=	Igual a
	< >	Distinto de
	<	Menor que
	>	Mayor que
	>=	Mayor o igual a
	<=	Menor o igual a
Lógicos	AND	Y lógico
	NOT	Negación
	OR	O lógico
Concatenación	| |	Concatenación

Tabla: Operadores en PL/SQL.

VÍDEO DE INTERÉS

Visualiza los distintos cursos sobre el lenguaje PL-SQL:

5. ESTRUCTURAS DE CONTROL

Una vez que ya sabes lo básico sobre Oracle y PL/SQL, es el momento de expliques a tu ayudante cuáles son y en qué se diferencian las diferentes estructuras de control. Para ello, crearéis tanto estructuras de control de flujo como estructuras repetitivas sobre una base de datos obsoleta, introduciendo la sintaxis IF y LOOP entre otras.

Las estructuras de control, al igual que en la mayoría de lenguajes de programación, son las de control de flujo y las repetitivas.

5.1 Estructuras de control de flujo

En los procedimientos almacenados solo se dispone de la estructura condicional IF. Su sintaxis se muestra a continuación:

```
DECLARE
-- Sección para declaración de variables
BEGIN
IF (expresión) THEN
-- Instrucciones
ELSIF (expresión) THEN
-- Instrucciones
ELSE
-- Instrucciones
END IF;
END;
```

Un aspecto importante es que la instrucción condicional anidada es ELSIF y no "ELSEIF".

5.2 Estructuras repetitivas

En los procedimientos almacenados se dispone de los siguientes iteradores o bucles: LOOP, WHILE y FOR.

LOOP: El bucle LOOP, se repite tantas veces como sea necesario hasta que se fuerza su salida con la instrucción EXIT. Su sintaxis es la siguiente:

```
DECLARE
-- Sección para declaración de variables
BEGIN
LOOP
   -- Instrucciones
   IF (expresión) THEN
      -- Instrucciones
      EXIT;
   END IF;
END LOOP;
END;
```

EJEMPLO PRÁCTICO

Trabajas para una empresa que se dedica al desarrollo de aplicaciones matemáticas para niños y, mediante un LOOP, debes implementar un algoritmo que muestre la tabla de multiplicar de un número dado. ¿Cómo lo harías?

SOLUCIÓN

```
DECLARE
                        MULTIPLICANDO NUMBER := 7;
                        MULTIPLICADOR NUMBER := 1;
                        RESULTADO NUMBER;
                      BEGIN
                        LOOP
                         -- SALIR DEL BUCLE CUANDO MULTIPLICADOR ES
MAYOR QUE 10
                          EXIT WHEN MULTIPLICADOR > 10;
                          RESULTADO := MULTIPLICANDO *
MULTIPLICADOR;
                            DBMS_OUTPUT.PUT_LINE(MULTIPLICANDO || '
X ' || MULTIPLICADOR || ' = ' || RESULTADO);
                          MULTIPLICADOR := MULTIPLICADOR + 1;
                        END LOOP;
                       END;
```

WHILE: El bucle WHILE, se repite mientras que se cumpla expresión.

```
DECLARE
-- Sección para declaración de variables
BEGIN

WHILE (expresión) LOOP
    -- Instrucciones
END LOOP;
END;
```

EJEMPLO PRÁCTICO

Trabajas en el departamento de informática de una empresa de formación y, para el próximo curso, se han matriculado 50 estudiantes. Debes realizar un algoritmo que inserte automáticamente el Id de estos 50 alumnos en la base de datos ¿Cómo lo harías usando la estructura repetitiva WHILE? Nota: tomar como primer valor de inserción el 1 e ir aumentando en cada iteración.

Solución:

```
                    DECLARE
                V_contador NUMBER:=1
            BEGIN
               WHILE v_contador <= 50 LOOP
                  INSERT INTO Estudiante (Id) VALUES (v_contador)
                 V_contador:=v_contador + 1;
               END LOOP;
            END;
```

Cabe señalar que, para definir variables, podemos añadir la sección **DECLARE**, como se puede ver en el ejemplo.

FOR: Se repite tantas veces como se le indique en los identificadores inicio y final. En el caso de especificar REVERSE el bucle se recorre en sentido inverso.

```
FOR contador IN [REVERSE] inicio..final LOOP
    -- Instrucciones
END LOOP;
```

EJEMPLO PRÁCTICO

Implementa un algoritmo que mediante el uso de FOR muestre los números pares entre 1 y 10.

SOLUCIÓN

```
BEGIN
    FOR i IN 1 .. 10 LOOP
        IF (i MOD 2 = 0) THEN
                            dbms_output.put_line(i);
                        END IF;
                    END LOOP;
                END;
```

PARA SABER MÁS

Para saber más sobre las estructuras de programación de PL/SQL, consúltalas:

VÍDEO DE INTERÉS

Para profundizar sobre el funcionamiento de PL/SQL, visualiza el vídeo:

6. PROCEDIMIENTOS ALMACENADOS

Para esta antigua aplicación en la que estás trabajando junto a tu ayudante, Paula te ha pedido que crees un procedimiento almacenado que sirva para dar de alta un nuevo producto y que muestre un mensaje en caso de que el producto ya exista.

Un procedimiento es un programa que ejecuta una acción específica y que no devuelve ningún valor. Un procedimiento tiene un nombre, un conjunto de parámetros (opcional) y un bloque de código.

6.1 Sintaxis

La sintaxis de un procedimiento almacenado sigue la siguiente estructura:

```
CREATE [OR REPLACE] PROCEDURE <proc_name> [(<param1> [IN|OUT|IN OUT]
<type>,
    <param2> [IN|OUT|IN OUT] <type>,
    ...)]
AS -- declaración de variables locales
BEGIN -- Sentencias
[EXCEPTION] -- Sentencias control de excepción
END [<procedure_name>];
```

6.2 Parámetros

Se debe especificar el tipo de datos de cada parámetro. Al especificar el tipo de dato del parámetro no se debe especificar la longitud del tipo. Los parámetros pueden ser de entrada (IN), de salida (OUT) o de entrada salida (IN OUT). El valor por defecto es IN, y se toma ese valor en caso de que no se especifique nada.

```
CREATE OR REPLACE PROCEDURE Actualiza_Saldo(cuenta IN NUMBER,
new_saldo OUT NUMBER)
AS
  -- declaración de variables locales
BEGIN
  -- Sentencias
  UPDATE SALDOS_CUENTAS
  SET SALDO = new_saldo
  WHERE CO_CUENTA = cuenta;
END Actualiza_Saldo;
```

También se puede asignar un valor por defecto a los parámetros, utilizando la cláusula DEFAULT o el operador de asignación (:=).

```
CREATE OR REPLACE PROCEDURE Actualiza_Saldo(cuenta NUMBER, saldo
NUMBER DEFAULT 10)
AS
    -- declaración de variables locales
BEGIN
  -- Sentencias
  UPDATE SALDOS_CUENTAS
  SET SALDO = new_saldo
  WHERE CO_CUENTA = cuenta;
END Actualiza_Saldo;
```

Una vez creado y compilado el procedimiento almacenado se puede ejecutar. Si el sistema indica que el procedimiento se ha creado con errores de compilación se pueden ver estos errores de compilación con la orden SHOW ERRORS.

Existen dos formas de pasar argumentos a un procedimiento almacenado a la hora de ejecutarlo (en realidad es válido para cualquier subprograma). Son:

- **Notación posicional**: Se pasan los valores de los parámetros en el mismo orden en que el PROCEDURE los define.

```
BEGIN
    Actualiza_Saldo(200501, 2500);
    COMMIT;
END;
```

- **Notación nominal**: Se pasan los valores en cualquier orden nombrando explícitamente el parámetro.

```
BEGIN
Actualiza_Saldo(cuenta => 200501, new_saldo => 2500);
COMMIT;
END;
```

A la hora de ejecutar un procedimiento, podemos hacerlo de dos maneras:

- Haciendo uso del nombre del procedimiento dentro de un bloque PL/SQL:

```
BEGIN
  Actualiza_Saldo (200501, 2500);
END;
```

- Mediante la sentencia **EXEC** de la siguiente manera:

```
exec Actualiza_Saldo (200501, 2500);
```

En el caso de que el procedimiento no reciba parámetros, podemos poner el nombre del procedimiento tanto con paréntesis como sin ellos, por ejemplo: `Actualiza_Saldo` o `Actualiza_Saldo()`.

Un procedimiento almacenado está compuesto por bloques, uno como mínimo.

6.3 Estructura de un Bloque

Los bloques presentan una estructura específica compuesta de tres partes bien diferenciadas:

- La **sección declarativa** en donde se declaran todas las constantes y variables que se van a utilizar en la ejecución del bloque.
- La **sección de ejecución** que incluye las instrucciones a ejecutar en el bloque.
- La **sección de excepciones** en donde se definen los manejadores de errores que soportará el bloque.

De las anteriores partes, únicamente la sección de ejecución es obligatoria, que quedaría delimitada entre las cláusulas BEGIN y END.

6.4 Borrar un procedimiento almacenado

Para borrar un procedimiento almacenado se utiliza la sentencia DROP PROCEDURE, cuya sintaxis es la siguiente:

```
DROP PROCEDURE [IF EXISTS] <proc_name>
```

De esta forma se borra del servidor la rutina especificada.

La cláusula IF EXISTS es una extensión de MySQL, y evita que ocurra un error si el procedimiento no existe. Se genera una advertencia que puede verse con SHOW WARNINGS.

6.5 Declaración de Variables

Una variable es un elemento que se utiliza para guardar o manipular información durante la ejecución de un programa.

En Oracle, una variable se declara asignándole un nombre o "identificador" seguido del tipo de valor que puede contener.

La sintaxis genérica para la declaración de constantes y variables es: nombre_variable [CONSTANT] <tipo_dato> [NOT NULL][:=valor_inicial].
Dónde:

- tipo_dato: es el tipo de dato que va a poder almacenar la variable, este puede ser cualquiera de los tipos soportados.

- La cláusula CONSTANT indica la definición de una constante cuyo valor no puede ser modificado. Se debe incluir la inicialización de la constante en su declaración.

- La cláusula NOT NULL impide que a una variable se le asigne el valor nulo, y por tanto debe inicializarse a un valor diferente de NULL. Las variables que no son inicializadas toman el valor inicial NULL.

Por ejemplo, para declarar la variable v_location, VARCHAR2(15), a la que se le asigna el valor "Granada": v_location VARCHAR2(15) := 'Granada';

Cabe mencionar que existen dos tipos de variables diferentes: las variables del **sistema** y las variables de **usuario**. Los ejemplos vistos anteriormente correspondían al segundo tipo.

Por otro lado, las variables del sistema (también conocidas como variables de **entorno**) en Oracle se refieren a los parámetros de configuración a nivel de sistema que afectan a la instancia de la base de datos en su conjunto. Estos parámetros se definen a nivel de sistema y afectan a todos los usuarios y sesiones que se conectan a la base de datos. Estos parámetros son administrados por el DBA (Administrador de Base de Datos) y pueden configurarse a través del archivo de parámetros de inicialización (init.ora o spfile.ora) o mediante comandos SQL. Algunos ejemplos de variables del sistema serían los siguientes:

- **ORACLE_HOME**: esta variable de entorno te dice la ruta dónde está instalado Oracle en el sistema.

- **TNS_ADMIN**: contiene información sobre cómo conectarte a diferentes bases de datos Oracle. Es como un mapa que te guía a las bases de datos.

EJEMPLO PRÁCTICO

En la empresa donde trabajas se te ha encargado crear un procedimiento almacenado que aumente en un 10% el salario de aquellos empleados cuyo salario esté por debajo de la media. ¿Cómo lo realizarías?

Solución:

```
CREATE OR REPLACE PROCEDURE Aumento IS
salario_med NUMBER;
BEGIN
    SELECT avg(salario) INTO salario_med FROM empleado;
    UPDATE empleado
        SET salario = salario * 1,1
        WHERE salario < salario  med;
END;
```

7. FUNCIONES Y FUNCIONES DEFINIDAS POR EL USUARIO

La empresa ha decidido dar un plus a finales de año a todos aquellos trabajadores que lleven más de cinco años trabajando en la empresa. Para saber quiénes son los empleados que obtendrán este plus, Paula te pide que desarrolles una función en la base de datos que, en función de la fecha de contratación, devuelva el número de años de antigüedad del empleado.

En Oracle y, en general en cualquier SGBD, existen dos tipos principales de funciones: las funciones incorporadas o predefinidas en el sistema y las funciones definidas por el usuario.

Las funciones incorporadas, también conocidas como funciones internas o predefinidas, son proporcionadas por Oracle como parte del conjunto estándar de funciones de la base de datos. Estas funciones están disponibles para su uso inmediato y abordan tareas comunes de procesamiento y manipulación de datos, como SUM, AVG, MAX, MIN, TO_CHAR, UPPER, LOWER, y muchas otras. Como podemos ver, muchas de ellas son iguales que las que se vieron en MySQL.

No es necesario definir estas funciones, ya que Oracle las incluye de manera predeterminada.

Las funciones definidas por el usuario son funciones personalizadas creadas por los usuarios para extender la funcionalidad de los SGBD. Estas funciones permiten a los usuarios implementar lógica específica y compleja para realizar operaciones personalizadas en la base de datos.

Tienen una estructura similar a los procedimientos almacenados. Al igual que estos, tienen una cabecera, la sección de declaración de variables y el propio cuerpo de la función. La principal diferencia entre las funciones definidas por el usuario y los procedimientos es que las funciones siempre devuelven un valor, para lo cual se usa la cláusula RETURN.

La sintaxis básica de una función es la siguiente:

```
CREATE [OR REPLACE] FUNCTION <func_name> [(<param1> [IN|OUT|IN OUT]
<type>,
    <param2> [IN|OUT|IN OUT] <type>,
    ...)]
RETURN tipo_dato_retorno
AS -- declaración de variables locales
BEGIN – Sentencias
RETURN VALOR
[EXCEPTION] -- Sentencias control de excepción
END [<func_name>];
```

PARA SABER MÁS

Profundiza sobre las funciones básicas en Oracle:

Para ejecutar esta función existen dos opciones. La primera de ellas es llamarla desde una consulta SQL, por ejemplo: `SELECT <func_name> ([param]) FROM` **DUAL,** donde DUAL es una tabla auxiliar que nos permite hacer una consulta en ORACLE cuando no necesitamos obtener valores de ninguna tabla.

La otra forma es crear un bloque PL/SQL, por ejemplo:

```
DECLARE
    v_total <type>;
begin
    v_total := <func_name>([param]);
    dbms_output.put_line('El resultado es: ' || v_total);
end;
```

EJEMPLO PRÁCTICO

Trabajas en el departamento de informática de una tienda Online y te han pedido que hagas una función que, dado el precio de un artículo pasado como parámetro, devuelva el precio incrementado en 10. ¿Cómo implementarías y ejecutarías esa función?

SOLUCIÓN

```
CREATE OR REPLACE FUNCTION incremento (valor IN NUMBER)
                  RETURN NUMBER IS
                  BEGIN
                     RETURN valor + 10;
                  END;
```

Para ejecutar esta función existen dos opciones. La primera de ellas es llamarla desde una consulta SQL:

```
SELECT incremento(10) FROM DUAL;
```

La otra forma es crear un bloque PL/SQL:

```
DECLARE
    v_total NUMBER :=10;
begin
    v_total := incremento(v_total);
    dbms_output.put_line('El resultado de sumar 10 es: ' ||
v_total);
end;
```

8. EVENTOS: DISPARADORES O TRIGGERS

A día de hoy, cada vez que la empresa para la que trabajas vende alguno de sus productos, alguien debe introducir manualmente el número de existencias que quedan en stock. Paula te pide que implementes un TRIGGER que se encargue de automatizar este proceso.

Un evento en una base de datos generalmente se refiere a una acción que ocurre en un momento específico, ya sea en una fecha y hora programada o tras una acción específica sobre la base de datos. Algunos ejemplos de eventos programables incluyen la ejecución de copias de seguridad programadas, la actualización de estadísticas de la base de datos, la ejecución de tareas de mantenimiento, etc.

El ejemplo más tipo de evento en una base de datos es el trigger o disparador. Un trigger es un objeto que se asocia a una tabla y se ejecuta automáticamente cuando ocurre un evento específico en dicha tabla. Los triggers son utilizados para implementar lógica adicional o realizar acciones especiales en respuesta a eventos, como la inserción, actualización o eliminación de datos en una tabla. Cuando se produce el evento especificado, el trigger se activa y ejecuta su lógica. Esta lógica puede incluir actualizaciones en otras tablas, generación de registros de auditoría o cualquier otra operación que sea necesaria en respuesta al evento.

La estructura básica de un trigger es la siguiente:

```
CREATE OR REPLACE TRIGGER nombre_trigger
[BEFORE | AFTER]
[INSERT | DELETE | UPDATE {OF columns} ON nombre_tabla]
   [REFERENCING OLD AS old NEW AS new]
   [FOR EACH ROW]
   [WHEN (condición)]
   [DECLARE]
   -- Declaraciones de variables locales
   BEGIN
   -- Lógica del trigger
   -- Sentencias SQL y/o llamadas a procedimientos almacenados
   [EXCEPTION]
   -- Manejo de excepciones
END;
```

Las cláusulas **BEFORE y AFTER** hacen referencia al momento en el que debe ejecutarse el disparador. BEFORE indica que el trigger debe ejecutarse antes de realizar la operación sobre la tabla. Con AFTER, el trigger se ejecutará después de realizar la operación pertinente sobre la tabla.

Las cláusulas **INSERT, UPDATE Y DELETE** hacen referencia a la operación que desencadenará la ejecución del trigger. Esta operación puede ser una inserción, un borrado o una modificación. En el caso de que queramos crear el trigger cuando se produce una modificación, también podemos especificar sobre qué campo de la tabla se debe hacer la modificación. Por ejemplo, si tuviéramos este trozo de código:

```
CREATE OR REPLACE TRIGGER trg_historial_inventario
BEFORE UPDATE OF cantidad_actual ON productos
```

El trigger solo se ejecutará cuando actualicemos el campo cantidad actual de la tabla productos. Si no se especifica ningún campo, el trigger se ejecutará cuando se realice cualquier modificación sobre la tabla.

[REFERENCING OLD AS old NEW AS new]: opcionalmente, permite hacer referencia a los valores antiguos (OLD) y nuevos (NEW) de los datos que activaron el trigger.

[FOR EACH ROW]: de forma opcional, indica que el trigger se ejecutará para cada fila afectada por el evento desencadenante.

[WHEN (condición)]: opcionalmente, especifica una condición que debe cumplirse para que el trigger se active. La condición puede ser una expresión booleana basada en los valores de la fila.

Ahora conoceremos los operadores OLD y NEW:

Cuando se realiza una operación de inserción (INSERT), actualización (UPDATE) o eliminación (DELETE) en una tabla, los valores antiguos y nuevos representan los valores antes y después de la operación en esa fila particular.

- **OLD**: hace referencia a los valores antiguos de la fila antes de la operación que activó el trigger. Solo está disponible en los triggers de actualización (UPDATE) y eliminación (DELETE), ya que en estos casos hay valores antiguos que se pueden obtener. La referencia a OLD se utiliza para acceder a los valores de las columnas antes de la operación.

- **NEW**: hace referencia a los valores nuevos de la fila después de la operación que activó el trigger. Está disponible en los triggers de inserción (INSERT) y actualización (UPDATE), ya que en estos casos hay valores nuevos que se pueden obtener. La referencia a NEW se utiliza para acceder a los valores de las columnas después de la operación.

Al utilizar OLD y NEW, puedes acceder a los valores de las columnas específicas en el trigger y utilizarlos en la lógica del mismo. Por ejemplo:

```
CREATE OR REPLACE TRIGGER trigger_ejemplo
AFTER INSERT OR UPDATE ON tabla_ejemplo
FOR EACH ROW
BEGIN
   IF INSERTING THEN
      DBMS_OUTPUT.PUT_LINE('Fila insertada: ' || :new.codigo);
   ELSIF UPDATING THEN
      IF :old.codigo <> :new.codigo THEN
         DBMS_OUTPUT.PUT_LINE('Valor del código ha cambiado de ' ||
:old.codigo || ' a ' || :new.codigo);
      END IF;
   END IF;
 END;
```

EJEMPLO PRÁCTICO

La empresa para que la trabajas necesita, a partir de hoy, guardar el historial de los salarios de todos los empleados. Tu responsable te indica que la mejor manera de hacerlo sería usando un Trigger. ¿Cómo lo harías?

SOLUCIÓN

```
CREATE OR REPLACE TRIGGER trg_historial_salarios
AFTER UPDATE OF salario ON empleados
FOR EACH ROW
BEGIN
  IF :OLD.salario != :NEW.salario THEN
    INSERT INTO historial_salarios (id_empleado, salario_anterior,
salario_nuevo, fecha_actualizacion)
    VALUES (:OLD.id_empleado, :OLD.salario, :NEW.salario, SYSDATE);
  END IF;
END trg_historial_salarios;
```

PARA SABER MÁS

Conoce más sobre los triggers:

ENLACE DE INTERÉS

Comprende más a fondo los conceptos de los disparadores, así como de los operadores old y new.

9. CURSORES

Ahora que tu ayudante ha alcanzado un nivel básico, consideras que sería bueno para su formación que investigase qué son los cursores y si pudiesen ser de utilidad en vuestro trabajo.

Un cursor es un objeto de base de datos que permite recorrer y manipular el resultado de una consulta SQL. Son especialmente útiles cuando se necesita manipular filas una por una en un bloque de código, como en procedimientos almacenados, funciones o triggers.

Principalmente, existen dos tipos de cursores:

- **Cursores implícitos**: se utilizan cuando la consulta SQL devuelve un solo registro. Este tipo de cursores se usan con la cláusula SELECT INTO.

```
DECLARE
   var_id Empleados.Id%TYPE;
   var_salario Empleados.Salario%TYPE;
BEGIN
   SELECT Id, Salario INTO var_id, var_salario
   FROM Empleados
   WHERE Ciudad = 'Madrid';

   DBMS_OUTPUT.PUT_LINE('Id: ' || var_id || ', Salario: ' ||
var_salario);
END;
```

```
%TYPE quiere decir que la variable definida será del mismo tipo de
dato que el campo al que está haciendo referencia. En el caso de la
línea: var_id Empleados.Id%TYPE;

La variable var_id será del mismo tipo que el campo Id de la tabla
Empleados.
```

- **Cursores explícitos**: son declarados y utilizados por el programador. Se utilizan cuando se requiere un mayor control sobre el procesamiento de las filas y se pueden utilizar en situaciones más complejas. Los cursores explícitos se definen mediante una declaración DECLARE y se manipulan utilizando sentencias OPEN, FETCH y CLOSE. Al utilizar un cursor explícito, se pueden realizar las siguientes acciones:

- **Abrir el cursor (OPEN)**: asocia el cursor con un conjunto de resultados y lo coloca en la primera fila.

- **Recuperar filas del cursor (FETCH)**: obtiene una fila del cursor actual y avanza a la siguiente.

- **Cerrar el cursor (CLOSE)**: libera los recursos asociados con el cursor.

```
DECLARE
   CURSOR cursor_ejemplo IS
      SELECT Id, Salario
      FROM Empleados;
      var_id Empleados.Id%TYPE;
      var_salario Empleados.Salario%TYPE;
BEGIN
   OPEN cursor_ejemplo;
   LOOP
     FETCH cursor_ejemplo INTO var_id, var_salario;
     EXIT WHEN cursor_ejemplo%NOTFOUND;
     DBMS_OUTPUT.PUT_LINE('Id: ' || var_id || ', Salario: '||
var_salario);
   END LOOP;
   CLOSE cursor_ejemplo;
END;
```

La instrucción **EXIT WHEN cursor_ejemplo%NOTFOUND** es una cláusula utilizada para salir de un bucle cuando no hay más filas disponibles en el cursor.

Otra forma de recorrer un cursor es mediante el uso del bucle FOR, el cual maneja automáticamente la apertura, el fetch y el cierre del mismo. Siguiendo con el ejemplo anterior, si usáramos en bucle FOR en lugar de un LOOP, quedaría de la siguiente manera:

```
DECLARE
   CURSOR cursor_ejemplo IS
      SELECT Id, Salario
      FROM Empleados;
BEGIN
   FOR rec IN cursor_ejemplo
   LOOP
      DBMS_OUTPUT.PUT_LINE('Id: ' || rec.Id || ', Salario: ' ||
rec.Salario);
   END LOOP;
END;
```

Lo que se hace es definir dentro del bucle la variable *rec,* que será la que usaremos para acceder a los diferentes valores de los campos Id y Salario en cada iteración.

EJEMPLO PRÁCTICO

Los dueños de la empresa para la que trabajas necesitan un informe que les muestre el id, nombre y salario de todos los empleados de la empresa. ¿Cómo lo harías si tuvieras que usar un cursor y recorrer dicho cursor con un LOOP?

SOLUCIÓN

```
DECLARE
  -- Declarar variables para el cursor
  v_id_empleado empleados.id_empleado%TYPE;
  v_nombre empleados.nombre%TYPE;
  v_salario empleados.salario%TYPE;

  -- Declarar el cursor
  CURSOR c_empleados IS
    SELECT id_empleado, nombre, salario
    FROM empleados;
BEGIN
  -- Abrir el cursor
  OPEN c_empleados;

  -- Recorrer el cursor y mostrar detalles de cada empleado
  LOOP
    FETCH c_empleados INTO v_id_empleado, v_nombre, v_salario;
    EXIT WHEN c_empleados%NOTFOUND;

    -- Procesar la fila (en este ejemplo, imprimir detalles)
    DBMS_OUTPUT.PUT_LINE('ID: ' || v_id_empleado || ', Nombre: ' ||
v_nombre || ', Salario: ' || v_salario);
  END LOOP;

  -- Cerrar el cursor
  CLOSE c_empleados;
END;
```

EJEMPLO PRÁCTICO

¿Cómo harías lo mismo que en el caso anterior, pero esta vez, utilizando un FOR para recorrer el cursor?

SOLUCIÓN

```
DECLARE
  -- Declarar variables para el cursor
  v_id_empleado empleados.id_empleado%TYPE;
  v_nombre empleados.nombre%TYPE;
  v_salario empleados.salario%TYPE;

  -- Declarar el cursor
  CURSOR c_empleados IS
    SELECT id_empleado, nombre, salario
    FROM empleados;
BEGIN
  -- Recorrer el cursor usando un bucle FOR
  FOR empleado IN c_empleados
  LOOP
    -- Obtener datos de la fila actual
    v_id_empleado := empleado.id_empleado;
    v_nombre := empleado.nombre;
    v_salario := empleado.salario;

    -- Procesar la fila (en este ejemplo, imprimir detalles)
    DBMS_OUTPUT.PUT_LINE('ID: ' || v_id_empleado || ', Nombre: ' ||
v_nombre || ', Salario: ' || v_salario);
  END FOR;
END;
```

PARA SABER MÁS

Conoce más características sobre los cursores:

ENLACE DE INTERÉS

También obtendrás más información sobre los cursores aquí:

10. EXCEPCIONES

Paula te ha pedido que implementes un procedimiento almacenado que sirva para validar el proceso de login de los usuarios en la aplicación. Pensando en ello, se te ocurre que quizá exista algún tipo de excepción que te facilite el trabajo.

En Oracle, las excepciones se utilizan para manejar errores y condiciones excepcionales que pueden surgir durante la ejecución de un bloque de código PL/SQL. Cuando se produce una excepción, el control se transfiere a una sección de manejo de excepciones donde se puede realizar un procesamiento especializado o se puede volver a lanzar la excepción.

Existen dos tipos de excepciones: las predefinidas y las definidas por el usuario.

- **Predefinidas:** son aquellas que se ejecutan automáticamente cuando se produce un error determinado. Algunos ejemplos son:
    - **too_many_rows:** cuando un SELECT INTO devuelve más de un registro.
    - **no_data_found:** cuando un SELECT INTO no devuelve ningún valor.
    - **dupval_on_index:** cuando se intenta insertar un valor repetido en una clave primaria o en un campo definido como UNIQUE.
    - **When_others:** se tratan cualquier tipo de excepción que no se haya indicado explícitamente.

```
BEGIN
  -- Bloque de código
EXCEPTION
  WHEN NO_DATA_FOUND THEN
      -- Manejo para cuando no se encuentran datos
  WHEN TOO_MANY_ROWS THEN
      -- Manejo para cuando se encuentran demasiadas filas
  WHEN DUP_VAL_ON_INDEX THEN
      -- Manejo para cuando se intenta insertar valores repetidos
  WHEN OTHERS THEN
      -- Manejo para cualquier otra excepción
END;
```

EJEMPLO PRÁCTICO

La compañía para la que trabajas necesita automatizar el alta de nuevos empleados, teniendo siempre en cuenta que se debe comprobar que dicho empleado no esté ya dado de alta. Sabiendo que debes usar las excepciones para resolverlo y que la PK es el DNI, ¿cómo lo harías?

SOLUCIÓN

```
DECLARE
  v_dni_empleado empleados.dni%TYPE := 123456789;
  v_nombre_empleado empleados.nombre%TYPE := 'Pepe Pérez';
  v_salario_empleado empleados.salario%TYPE := 60000;
BEGIN
  INSERT INTO empleados VALUES (v_dni_empleado, v_nombre_empleado,
v_salario_empleado);
  DBMS_OUTPUT.PUT_LINE('Empleado insertado correctamente.');
  EXCEPTION
    WHEN DUP_VAL_ON_INDEX THEN
      DBMS_OUTPUT.PUT_LINE('Error: El DNI del empleado ya existe en
la tabla.');
  END;
END;
```

- **Definidas por el usuario:** son aquellas definidas por el propio usuario. Para ello, se deben seguir tres pasos:

    - **Definición:** en la sección DECLARE de la siguiente manera: **nombre_excepcion EXCEPTION**.

    - **Lanzamiento:** lanzar la excepción mediante la cláusula **RAISE.**

    - **Tratamiento:** tratar la excepción en el apartado **EXCEPTION.**

A continuación, se muestra un ejemplo de cómo lanzar una excepción que controla que un importe no exceda de 50:

```
DECLARE
      ImporteIncorrecto EXCEPTION;
BEGIN
      IF Importe > 50 THEN
         RAISE ImporteIncorrecto;
      END IF;
      EXCEPTION
         WHEN ImporteIncorrecto THEN
           -- Manejo para la excepción personalizada
         WHEN OTHERS THEN
           -- Manejo para el resto de excepciones
END;
```

Si, por ejemplo, quisiéramos controlar la entrada de los menores de edad en un bar:

```
DECLARE
     mayor_edad EXCEPTION;
     v_age NUMBER := 15;
BEGIN
    IF v_age < 18 THEN
        RAISE mayor_edad;
    END IF;
    EXCEPTION
      WHEN mayor_edad THEN
         DBMS_OUTPUT.PUT_LINE('El usuario es menor de  edad. No
se permite el acceso.');
END;
```

EJEMPLO PRÁCTICO

En tu empresa habéis detectado que se han cometido errores al dar de alta algunos empleados, que han sido registrados con un salario negativo. ¿Cómo implementarías un procedimiento con una excepción personalizada para que esto no volviera a pasar?

SOLUCIÓN

```
CREATE OR REPLACE PROCEDURE insertar_empleado(p_id IN NUMBER,
p_nombre IN VARCHAR2, p_salario IN NUMBER) IS
  -- Declarar la excepción personalizada
  salario_negativo_exception EXCEPTION;

BEGIN
  -- Verificar si el salario es negativo
  IF p_salario < 0 THEN
      -- Lanzar la excepción personalizada
      RAISE salario_negativo_exception;
  END IF;

  -- Insertar el empleado en la tabla de empleados
  INSERT INTO empleados (id, nombre, salario)
  VALUES (p_id, p_nombre, p_salario);

  -- Confirmar que se ha insertado el empleado
  DBMS_OUTPUT.PUT_LINE('Empleado insertado: ' || p_nombre || ',
Salario: ' || p_salario);
EXCEPTION
 -- Manejar la excepción personalizada
WHEN salario_negativo_exception THEN
  DBMS_OUTPUT.PUT_LINE('Error: No se permite un salario negativo');
WHEN OTHERS THEN
  DBMS_OUTPUT.PUT_LINE('Error inesperado');
END;
```

EJEMPLO PRÁCTICO

Trabajas en el departamento de informática de un banco y debéis controlar que nadie saque por el cajero más dinero del que tiene en la cuenta. ¿Cómo lo harías?

SOLUCIÓN

```
CREATE OR REPLACE PROCEDURE realizar_retiro(
    p_cuenta_id NUMBER,
    p_monto NUMBER
)
IS
    saldo_insuficiente_exception EXCEPTION;
    saldo_actual NUMBER;
BEGIN
    SELECT saldo INTO saldo_actual
    FROM cuentas
    WHERE cuenta_id = p_cuenta_id;

    IF saldo_actual < p_monto THEN
        -- Lanzar la excepción personalizada
        RAISE saldo_insuficiente_exception;
    END IF;

    -- Realizar la actualización del saldo
    UPDATE cuentas
    SET saldo = saldo - p_monto
    WHERE cuenta_id = p_cuenta_id;

    DBMS_OUTPUT.PUT_LINE('Retiro de ' || p_monto || ' realizado en
la cuenta ' || p_cuenta_id);
EXCEPTION
    -- Manejar la excepción personalizada
    WHEN saldo_insuficiente_exception THEN
       DBMS_OUTPUT.PUT_LINE('Error: Saldo insuficiente');
    WHEN OTHERS THEN
       DBMS_OUTPUT.PUT_LINE('Error inesperado');
END;
```

PARA SABER MÁS

Comprueba más información sobre las excepciones en Oracle:

ENLACE DE INTERÉS

Las excepciones también pueden ayudar a detectar y tratar errores en tiempo de ejecución. Consulta más sobre esto:

11. AUTOMATIZACIÓN DE TAREAS

Llegados a este punto, ya eres capaz de crear tus propias funciones y procedimientos y, al implementarlos, has detectado que algunos de ellos deben ejecutarse periódicamente. Para no tener que estar pendiente de lanzarlas manualmente, te dispones a crear un evento para que dichas funciones y procedimientos se ejecuten automáticamente.

Una potente herramienta que Oracle nos ofrece para la automatización de tareas es **DBMS_SCHEDULER,** la cual permite programar y automatizar tareas y trabajos en la base de datos. Proporciona una manera flexible y versátil de gestionar tareas programadas, y se utiliza para realizar una variedad de funciones, como ejecutar procedimientos, enviar notificaciones, realizar copias de seguridad y muchas otras tareas relacionadas con la administración de la base de datos.

Una de las principales utilidades que ofrece esta herramienta es la creación de trabajos o **jobs**, los cuales podremos ejecutar periódicamente en el intervalo que deseemos. La estructura básica para la creación de un **job** sería la siguiente:

```
BEGIN
  DBMS_SCHEDULER.create_job (
    job_name  => 'NOMBRE_DEL_JOB',  -- Nombre del job
    job_type  => 'PLSQL_BLOCK',   -- Tipo de job (puede ser
PLSQL_BLOCK,          PLSQL_SCRIPT, etc.)
    job_action => 'BEGIN TUS_ACCIONES_PLSQL; END;',-- Acción a
realizar
    start_date => SYSTIMESTAMP, -- Fecha y hora de inicio
    repeat_interval => 'FREQ=HOURLY; INTERVAL=1',-- Intervalo de
repetición
    (los valores para el parámetro FREQ pueden ser: ONCE (una sola
vez), DAILY, WEEKLY, MONTHLY, HOURLY, MINUTELY, YEARLY)
    enabled   => TRUE, -- Habilitar el job (TRUE o FALSE)
    auto_drop => FALSE -- No eliminar automáticamente el job después
de la ejecución  );
END;
```

Una vez creado el job, podemos tanto parar su ejecución como eliminarlo. Las sentencias para ello serían las siguientes:

```
BEGIN
  DBMS_SCHEDULER.stop_job('NOMBRE_DEL_JOB', TRUE); -- Detiene la
ejecución
END;

BEGIN
  DBMS_SCHEDULER.drop_job('NOMBRE_DEL_JOB', FORCE => TRUE); -- Borra
el job
END;
```

EJEMPLO PRÁCTICO

En la base de datos con la que trabajáis en tu empresa, existe un procedimiento almacenado llamado insertaHora, el cual inserta la hora del sistema en una tabla. ¿Sabrías crear un job llamado MI_JOB_HORA que ejecute este procedimiento cada hora a partir del momento actual?

SOLUCIÓN

```
BEGIN
   DBMS_SCHEDULER.create_job (
   job_name        => 'MI_JOB_HORA',
   job_type        => 'PLSQL_BLOCK',
   job_action      => 'BEGIN insertaHora; END;',
   start_date      => SYSTIMESTAMP,
   repeat_interval => 'FREQ=HOURLY; INTERVAL=1',
   enabled         => TRUE,
   auto_drop       => FALSE
   );
      END;
```

EJEMPLO PRÁCTICO

Ya has establecido el procedimiento, pero la empresa decide que ya no es necesario mantener esta funcionalidad. Debes detener la ejecución del Job y borrarlo. ¿Cómo lo harías?

SOLUCIÓN

```
BEGIN
   DBMS_SCHEDULER.stop_job('MI_JOB_HORA', TRUE);
   DBMS_SCHEDULER.drop_job('MI_JOB_HORA',FORCE =>
TRUE);
END;
```

RESUMEN FINAL

A lo largo de la unidad hemos desglosado las características generales de Oracle, desde tipos de datos y operadores hasta estructuras de control y procedimientos almacenados.

Además, se ha visto una introducción a la sintaxis de PL/SQL, la gestión de procedimientos almacenados y funciones, así como la implementación de desencadenadores para manejar eventos específicos en la base de datos.

Al mismo tiempo, se han tratado conceptos clave como cursores y excepciones, así como la automatización de tareas para mejorar la eficiencia operativa.