UNIDAD 6: PROGRAMACIÓN DE BASES DE DATOS

Módulo Profesional: Bases de Datos

Índice

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. Por último, 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.

INTRODUCCIÓN

Son muchas las tareas que se pueden automatizar dentro de una base de datos mediante el uso de un lenguaje de programación. Incluso se pueden programar algunas restricciones a la hora de manipular los datos. Para ello existe un lenguaje de programación llamado PL-SQL y las empresas del sector lo utilizan ya que ofrece varias ventajas como, por ejemplo: manejar variables de diferentes tipos, facilitar el uso de estructuras de control de flujo y controlar excepciones, entre otras acciones. Incluir estos elementos son parte esencial en cualquier lenguaje de programación.

CASO INTRODUCTORIO

Un centro educativo cuyo director es amigo íntimo de Paula requiere actualizar su base de datos y crear una serie de acciones que le permitan trabajar con los datos que tiene almacenados. Concretamente, se tienen 2 tablas diferentes para almacenar alumnos; en una de las tablas se almacenarán los mayores de 18 años y en otra los menores de 18 años. Para automatizar esta acción es importante que el equipo de Paula sepa crear un procedimiento almacenado implementando la sintaxis correcta. De esta manera, cada vez que se matricule un alumno, sus datos serán almacenados de forma automática en una tabla u otra, dependiendo de la edad. Sin un procedimiento almacenado sería una ardua tarea para el departamento de administración tener que ir cambiando de tabla cada vez que un alumno se matricule, teniendo presente en todo momento su fecha de nacimiento.

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

1. INTRODUCCIÓN Y CARACTERÍSTICAS

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

Cómo instalar ORACLE DATABASE 11G EXPRESS EDITION:

https://www.youtube.com/watch?v=gfUQ-cBDKxA

Para descargar PL/SQL Developer:

https://www.filehorse.com/es/descargar-oracle-sql-developer-64/descargar/

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**: Es un símbolo simple o compuesto que tiene una función especial en los procedimientos almacenados. Estos pueden ser: Operadores Aritméticos, Operadores Lógicos y Operadores Relacionales.
- **Identificador**: Son empleados para nombrar objetos de programas, así como a unidades dentro del mismo. 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:

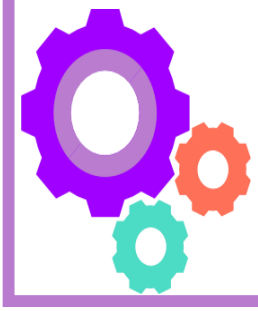
```
-- línea simple
/* Conjunto de Líneas
*/
```

2. TIPOS DE DATOS

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

En este enlace se pueden ver los tipos de datos en PL-SQL:

https://docs.oracle.com/es-ww/iaas/data-safe/doc/supported-data-types.html

ENLACE DE INTERÉS

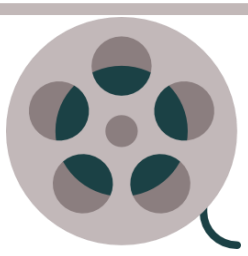
Es muy recomendable consultar el siguiente enlace que habla sobre el ámbito y visibilidad de las variables:

https://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=135:declaracion-de-variables-en-programacion-y-ambito-de-una-variable-global-o-local-visibilidad-cu00205a&catid=36&Itemid=60

3. OPERADORES

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



VIDEO DE INTERÉS

Se recomienda visitar el siguiente canal de YouTube donde se pueden visualizar distintos cursos sobre el lenguaje PL-SQL:

https://www.youtube.com/watch?v=l6wOghW_gNI&list=PL2Z95CSZ1N4EO3wqFmTBNZXCovLpxkEqB

4. ESTRUCTURAS DE CONTROL

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

Las **estructuras de control de flujo**, en los procedimientos almacenados solo se dispone de la estructura condicional IF. Su sintaxis se muestra a continuación:

```
IF (expresión) THEN
   -- Instrucciones
ELSIF (expresión) THEN
   -- Instrucciones
ELSE
   -- Instrucciones
END IF;
```

Es importante que la instrucción condicional anidada es ELSIF, no "ELSEIF".

Las 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:

```
LOOP
   --  Instrucciones
   IF (expresión) THEN
        -- Instrucciones
   EXIT;
   END IF;
END LOOP;

DECLARE
  contador NUMBER := 0;
BEGIN
  LOOP
    IF contador > 10 THEN
      EXIT;
    END IF;
     contador := contador + 1;
     IF (contador MOD 2 = 0) THEN
            dbms_output.put_line(contador);
     END IF;
  END LOOP;
```

```
END;
```

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

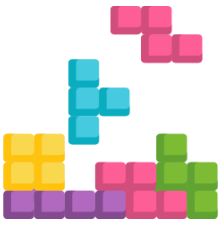
```
WHILE (expresión) LOOP
   -- Instrucciones
END LOOP;
```

- **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;
```

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

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



EJEMPLO PRÁCTICO

Hacer uso de una estructura repetitiva WHILE para insertar 50 estudiantes. 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
          VALUES (v_contador);
      v_contador:=v_contador+1;
  END LOOP;
END;
```

PARA SABER MÁS

Más ejemplos sobre bucles:

http://elbauldelprogramador.com/plsql-estructuras-basicas-de-control/

5. PROCEDIMIENTOS ALMACENADOS

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.

La **sintaxis** de un procedimiento almacenado sigue la siguiente estructura:

```
CREATE 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>];
```

En los **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 PROCEDURE Actualiza_Saldo(cuenta NUMBER,
new_saldo 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 PROCEDURE Actualiza_Saldo(cuenta NUMBER,
   new_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;
```

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

En cuanto a la **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.

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.

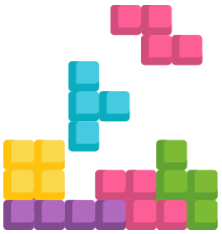
La cláusula IF EXISTS es una extensión de MySQL, y evita que ocurra un error si el procedimiento no existe. Se genera una advertencia que puede verse con SHOW WARNINGS.

En la **declaración de variables** se declaran las variables que va a necesitar el programa. Una variable se declara básicamente 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], donde:

- **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’;



EJEMPLO PRÁCTICO

Crear un procedimiento almacenado que aumente en un 10% el salario de aquellos empleados cuyo salario esté por debajo de la media.

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;
```

RESUMEN FINAL

El lenguaje PL/SQL permite ampliar funcionalidad que el lenguaje SQL no aporta. En este sentido, es posible crear procedimientos almacenados, funciones, manejar variables, trabajar con datos, control de excepciones, estructuras modulares, etc. Esta unidad pretende presentar al alumno unos conocimientos básicos sobre este lenguaje para poder adentrarse en él.