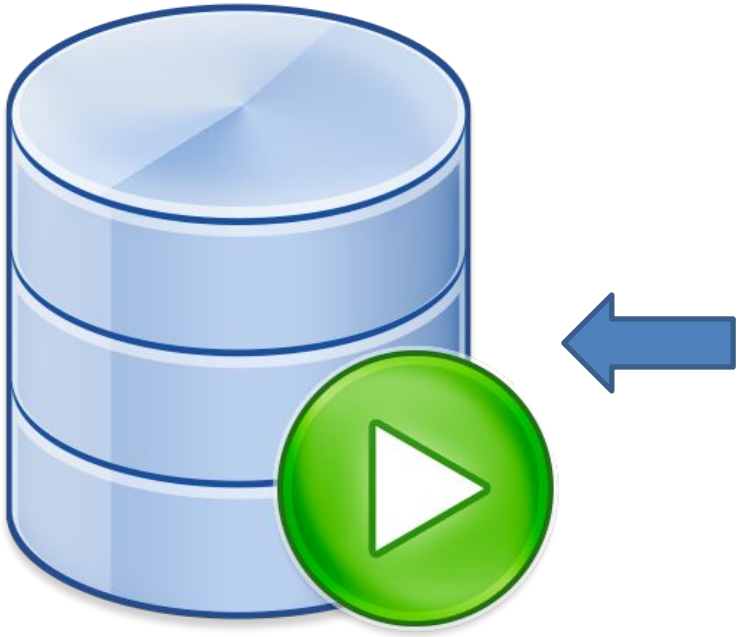
UD 6. PL/SQL



console [Oracle - @localhost]

Tx: Manual C##TESTER conso

```
1  create PROCEDURE simpleprocedure (inval NUMBER)
2  IS
3    tmpvar   NUMBER;
4    tmpvar2   NUMBER;
5    total     NUMBER;
6  BEGIN
7    tmpvar := 0;
8    tmpvar2 := 0;
9    total := 0;
10   FOR lcv IN 1 .. inval
11   LOOP
```

simpleprocedure()

Índice

- Introducción
- Estructuras de bloques
- Estructuras de Control
  - Ejecución selectiva: Condicionales
  - Ejecución repetitiva: Bucles
- **Procedimientos**
- **Funciones**
- Variables
- Registros
- Cursores
- Paquetes
- Disparadores (**Triggers**) y Eventos
- Gestión de excepciones

Introducción

PL/SQL (Programing Language/Structured Query Language) es un lenguaje de programación que soporta todas las sentencias de SQL e incluyendo nuevas características como el manejo de variables, estructuras modulares, …

En otros sistemas gestores de bases de datos existen otros lenguajes procedimentales:

- SQL Server utiliza **Transact SQL**
- PostgreSQL usa **PL/pgSQL**
- Informix usa **Informix 4GL**

Estructuras de Bloque

PL/SQL es un lenguaje **estructurado** en **bloques**, que a su vez pueden contener otros sub-bloques. Un **bloque** (o sub-bloque) permite **agrupar** en forma lógica un grupo de **sentencias**. De esta manera se pueden efectuar declaraciones de variables que sólo tendrán validez en los bloques donde éstas se definan.

Un bloque PL/SQL tiene tres partes:

- **DECLARE.** Sección de declaración. Define todas las variables, cursores, subprogramas y otros elementos que se utilizarán en el programa.
- **BEGIN y END**. Sección de ejecución. Incluye sentencias PL / SQL ejecutables del programa. Debe tener al menos una línea de código ejecutable, aunque sea NULL.
- **EXCEPTION**. Sección de manejo de excepciones.

Delimiter

Para ejecutar varias sentencias es necesario modificar temporalmente el carácter separador que se utiliza para delimitar las sentencias SQL.

El carácter separador que se utiliza por defecto en SQL es (;).

En los ejemplos vamos a utilizar los caracteres $$ para delimitar las instrucciones SQL, pero es posible utilizar cualquier otro carácter.

Ejemplo: configurar los caracteres $$ como los separadores entre las sentencias SQL.

DELIMITER $$

En este ejemplo volvemos a configurar que el carácter separador es el punto y coma.

DELIMITER ;

Estructuras de Bloque

Ejemplo

```
DELIMITER $$
BEGIN
    SHOW TABLES;
END $$
DELIMITER ;
```

NOTA

- En **MySQL** se usa **DELIMITER** para poder ejecutar código.
- En **Oracle** se usa la barra **/** para ejecutar el código.
- Si usas SQL*Plus deberás ejecutar al inicio de sesión la siguiente orden para que se habilite la salida:

  SET SERVEROUTPUT **ON**

Estructuras de Bloque

Sintaxis

```
[ DECLARE
      constantes,
      variables,
      cursores,
      excepciones definidas por el usuario
       ...
]
BEGIN
      Sentencias
[ EXCEPTION
      Acciones a realizar cuando se produce alguna excepción
]
END
```

Estructuras de Bloque

Ejemplo

```
DECLARE
    fecha DATE;
BEGIN
    SELECT CURDATE() INTO fecha FROM tabla;
END
```

Dentro de un bloque BEGIN … END la sentencia SELECT es:

SELECT campos **INTO** variable **...**

Estructuras de control

Instrucciones condicionales IF-THEN-ELSE

IF permite ejecutar una secuencia de acciones si se cumple una condición.

Existen tres modos para esta instrucción:

- IF – THEN
- IF – THEN – ELSE
- IF – THEN – ELSIF

Sintaxis

```
IF search_condition THEN statement_list
    [ELSEIF search_condition THEN statement_list] ...
    [ELSE statement_list]
END IF
```

Puede encontrar más información en la documentación oficial de MySQL.

Estructuras de control : *IF-THEN-ELSE*
Ejemplos

Ejemplo:

```
IF tipo_trans = 'CR' THEN
    UPDATE cuentas SET balance = balance + credito WHERE …
ELSE
    UPDATE cuentas SET balance = balance – debito WHERE …
END IF;
```

Estructuras de control : *IF-THEN-ELSE*
Ejemplos

Ejemplo:

```
BEGIN
 …
    IF  sueldo > 50000 THEN
          bonus : = 1500;
     ELSIF sueldo > 35000 THEN
          bonus : = 500;
     ELSE
          bonus : = 100;
     END IF;
     INSERT INTO sueldos VALUES (emp_id, bonus, );
END
```

Estructuras de control

Instrucciones condicionales CASE

Existen dos formas de utilizar CASE:

```
CASE case_value
    WHEN when_value THEN statement_list
    [WHEN when_value THEN statement_list] ...
    [ELSE statement_list]
END CASE
```

o

```
CASE WHEN search_condition THEN statement_list
    [WHEN search_condition THEN statement_list] ...
    [ELSE statement_list]
END CASE
```

Puede encontrar más información en la documentación oficial de MySQL.

Estructuras de control

Instrucciones repetitivas LOOP

LOOP: Crea un bucle, ejecuta las sentencias que están dentro del bucle una y otra vez hasta la palabra clave EXIT o END LOOP

Sintaxis:

```
LOOP
    secuencia_de_instrucciones
    IF condición THEN
        EXIT; -- Termina inmediatamente
    END IF;
END LOOP;
```

Ejemplo:

```
label1: LOOP
   SET p1 = p1 + 1;
   IF p1 < 10 THEN
    ITERATE label1;
   END IF;
   LEAVE label1;
END LOOP label1;
```

Puede encontrar más información en la documentación oficial de MySQL.

Estructuras de control

Instrucciones repetitivas WHILE

WHILE: Mientras se cumpla una condición, se ejecuta una secuencia de sentencias encerradas por las palabras clave WHILE LOOP y END LOOP.

Sintaxis:

```
WHILE condición DO
  secuencia_de_sentencias
END WHILE;
```

Ejemplo:

```
…
contador := 1;
WHILE contador < 100 DO
      contador := contador + 1;
END WHILE;
```

Puede encontrar más información en la documentación oficial de MySQL.

Estructuras de control

Instrucciones repetitivas: REPEAT

Sintaxis:

```
[begin_label:] REPEAT
    statement_list
UNTIL search_condition
END REPEAT [end_label]
```

Más información aquí.

Ejemplo:

```
DELIMITER $$
DROP PROCEDURE IF EXISTS ejemplo_bucle_repeat$$
CREATE PROCEDURE ejemplo_bucle_repeat(IN tope INT,
OUT suma INT)
BEGIN
  DECLARE contador INT;
       SET contador = 1;
SET suma = 0;
   REPEAT
      SET suma = suma + contador;
      SET contador = contador + 1;
      UNTIL contador > tope
   END REPEAT;
END
$$

DELIMITER ;
CALL ejemplo_bucle_repeat(10, @resultado);
SELECT @resultado;
```

Procedimientos

Procedimiento: conjunto de instrucciones SQL que se almacena asociado a una base de datos.

Es un objeto que se crea con la sentencia **CREATE PROCEDURE**.
Se invoca con la sentencia **CALL**.

Un procedimiento puede tener cero o muchos parámetros de entrada y cero o muchos parámetros de salida.

Procedimientos

Sintaxis:

```
CREATE
    [DEFINER = { user | CURRENT_USER }]
    PROCEDURE sp_name ([parametros [,...]])
    [caracteristicas...] sentencias
```

Parametros: [**IN** | **OUT** | INOUT] nombre_parametro **tipo** Caracteristicas:

COMMENT 'string' | LANGUAGE SQL | [**NOT**] DETERMINISTIC
| { **CONTAINS** SQL | **NO** SQL | **READS** SQL **DATA** | MODIFIES SQL **DATA** }
| SQL SECURITY { **DEFINER** | INVOKER }

Más información en la documentación oficial de MySQL.

Funciones

Función: conjunto de instrucciones SQL que se almacena asociado a una base de datos.

Es un objeto que se crea con la sentencia **CREATE FUNCTION**.
Se invoca con la sentencia **SELECT**.

Un procedimiento puede tener cero o muchos parámetros de entrada y siempre devuelve un valor, asociado al nombre de la función.

Funciones

Sintaxis:

```
CREATE
    [DEFINER = { user | CURRENT_USER }]
    FUNCTION sp_name ([parametros [,...]])
    RETURNS tipo
    [caracteristicas...] sentencias
```

Parametros: nombre_parametro **tipo**

Caracteristicas:

COMMENT 'string' | LANGUAGE SQL | [**NOT**] DETERMINISTIC
| { **CONTAINS** SQL | **NO** SQL | **READS** SQL **DATA** | MODIFIES SQL **DATA** }
| SQL SECURITY { **DEFINER** | INVOKER }

Más información en la documentación oficial de MySQL

Cursores

Los cursores permiten almacenar una conjunto de filas de una tabla en una estructura de datos que podemos ir recorriendo de forma secuencial.

Los cursores tienen las siguientes propiedades:

- ***Asensitive***: puede o no hacer una copia de su tabla de resultados.
- ***Read only***: son de sólo lectura. No permiten actualizar los datos.
- ***Nonscrollable***: sólo pueden ser recorridos en una dirección y no podemos saltarnos filas.

Cuando declaramos un cursor dentro de un procedimiento almacenado debe aparecer antes de las declaraciones de los manejadores de errores (HANDLER) y después de la declaración de variables locales.

Más información aquí.

Cursores

Operaciones con cursores

DECLARE: El primer paso es declarar el cursor.

Sintaxis:
```
DECLARE cursor_name CURSOR FOR select_statement
```

OPEN: abre el cursor.

Sintaxis:
```
OPEN cursor_name
```

FETCH: Después de abrir el cursor, se obtiene cada una de las filas con FETCH.

Sintaxis:
```
FETCH [[NEXT] FROM] cursor_name INTO var_name [, var_name] ...
```

Al recorrer un cursor y no quedar filas por recorrer se lanza el error **NOT FOUND**, que se corresponde con **SQLSTATE '02000'**. Por eso será necesario declarar un ***handler*** para manejar este error.

Sintaxis:
```
DECLARE CONTINUE HANDLER FOR NOT FOUND ...
```

CLOSE: Al terminar con un cursor tenemos que cerrarlo.

Sintaxis:
```
CLOSE cursor_name
```

Triggers

Un *trigger* o *disparador* es un objeto de la base de datos que está asociado con una tabla y que se activa cuando ocurre un evento sobre la tabla.

Los eventos que pueden ocurrir sobre la tabla son:

- **INSERT**: El *trigger* se activa cuando se inserta una nueva fila sobre la tabla asociada.
- **UPDATE**: El *trigger* se activa cuando se actualiza una fila sobre la tabla asociada.
- **DELETE**: El *trigger* se activa cuando se elimina una fila sobre la tabla asociada.

Triggers

Sintaxis:

```
CREATE
    [DEFINER = { user | CURRENT_USER }]
    TRIGGER nombre_trigger
    { BEFORE | AFTER }   { INSERT | UPDATE | DELETE }
    ON nombre_tabla FOR EACH ROW
    [{ FOLLOWS | PRECEDES }]
    sentencias_trigger
```

Más información en la documentación oficial de MySQL

Triggers
Ejemplo

Crea la **base de datos** *test_trigger* que contenga la **tabla** llamada *alumnos* con:

- id (entero sin signo)
- nombre (cadena de caracteres)
- apellido1 (cadena de caracteres)
- apellido2 (cadena de caracteres)
- nota (número real)

Una vez creada la tabla escriba **dos triggers** con las siguientes características:

Trigger 1: trigger_check_nota_before_insert

- Se ejecuta sobre la tabla alumnos.
- Se ejecuta *antes* de una operación de *inserción*.
- Si el nuevo valor de la nota que se quiere insertar es negativo, se guarda como 0.
- Si el nuevo valor de la nota que se quiere insertar es mayor que 10, se guarda como 10.

Triggers
Ejemplo

Trigger2 : trigger_check_nota_before_update

- Se ejecuta sobre la tabla alumnos.
- Se ejecuta antes de una operación de actualización.
- Si el nuevo valor de la nota que se quiere actualizar es negativo, se guarda como 0.
- Si el nuevo valor de la nota que se quiere actualizar es mayor que 10, se guarda como 10.

Una vez creados los triggers escriba varias sentencias de inserción y actualización sobre la tabla alumnos y verifica que los *triggers* se están ejecutando correctamente.

Gestión de excepciones

Sintaxis: **DECLARE ... HANDLER**

```
DECLARE accion HANDLER
 FOR condicion [, condicion2] …
   BEGIN
     Sentencias
   END;
```

Acciones:
CONTINUE: La ejecución continúa.
EXIT: Termina la ejecución del programa.
UNDO: No está soportado en MySQL.

Condiciones:
mysql_error_code
SQLSTATE [VALUE] sqlstate_value
condition_name
SQLWARNING
NOT FOUND
SQLEXCEPTION

Puede encontrar más información en la documentación oficial de MySQL.

Gestión de Excepciones

Ejemplo: Error 1051

Cuando se intenta acceder a una **tabla** que **no existe** en la base de datos, se produce el **error 1051** de MySQL.

El ejemplo declara un handler que se ejecutará con el error 1051. La acción del handler es CONTINUE, ejecutará las instrucciones especificadas en el cuerpo del handler el procedimiento almacenado continuará su ejecución.

```
DECLARE CONTINUE HANDLER FOR 1051
BEGIN
  -- body of handler
END;
```

Gestión de Excepciones

Ejemplos: SQLSTATE, SQLWARNING

Cuando se intenta acceder a una **tabla** que **no existe** en la base de datos, también se puede utilizar la variable SQLSTATE, **SQLSTATE es 42S02**.

```
DECLARE CONTINUE HANDLER FOR SQLSTATE '42S02'
BEGIN
  -- body of handler
END;
```

Ejemplo para **SQLWARNING**: Es equivalente a indicar todos los valores de SQLSTATE que empiezan con 01.

```
DECLARE CONTINUE HANDLER FOR SQLWARNING
BEGIN
  -- body of handler
END;
```

Gestión de Excepciones
Ejemplo: NOT FOUND, SQLEXCEPTION

Ejemplo de **NOT FOUND**:
Es equivalente a indicar todos los valores de **SQLSTATE** que empiezan con **02**.
Se utiliza con **cursores,** controlar qué ocurre cuando un cursor alcanza el final del data set. Si no hay más filas disponibles en el cursor, entonces ocurre una condición de NO DATA con un valor de SQLSTATE igual a 02000. Para detectar esta condición podemos usar un handler para controlarlo.

```
DECLARE CONTINUE HANDLER FOR NOT FOUND
BEGIN
  -- body of handler
END;
```

Ejemplo de **SQLEXCEPTION**::
Es equivalente a indicar todos los valores de SQLSTATE que empiezan por 00, 01 y 02.

```
DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
BEGIN
  -- body of handler
END;
```