UT10. SQL MODO PROGRAMACIÓN

Módulo: BASES DE DATOS

Curso 2022/2023. 1° DAM

Ruth Lospitao Ruiz

CONTENIDOS

- Introducción. Conceptos generales
- Tipos de datos, identificadores, variables.
- Operadores. Estructuras de control.
- Cursores.
- Procedimientos y funciones almacenados.
- Excepciones

INTRODUCCIÓN

¿POR QUÉ PL/SQL?

- PL/SQL es un lenguaje procedimental estructurado en bloques que amplía la funcionalidad de SQL.
- Con PL/SQL podemos usar sentencias SQL para manipular datos y sentencias de control de flujo para procesar los datos. Por tanto, **PL/SQL combina la potencia de SQL para la manipulación de datos, con la potencia de los lenguajes procedimentales para procesar los datos.**
- Aunque PL/SQL fue creado por Oracle, hoy día todos los gestores de bases de datos utilizan un lenguaje procedimental muy parecido al ideado por Oracle para poder programar las bases de datos

INTRODUCCIÓN. CONCEPTOS GENERALES DEL LENGUAJE DE PROGRAMACIÓN INTEGRADO EN EL SGBD.

- SQL (Structured Query Language o lenguaje de consulta estructurada) es un lenguaje estándar específico diseñado para administrar, y recuperar información de Sistemas Gestores de Bases de Datos Relacionales (SGBDR). Es un lenguaje declarativo.
- PL/SQL (Procedural Language/Structured Query Language) es un lenguaje de programación procedimental incrustado en Oracle, que extiende SQL, incluyendo nuevas características:
  - Manejo de variables.
  - Estructuras modulares.
  - Estructuras de control de flujo y toma de decisiones.
  - Control de excepciones.

INTRODUCCIÓN. CONCEPTOS GENERALES DEL LENGUAJE DE PROGRAMACIÓN INTEGRADO EN EL SGBD.

- Otros SGBDR también tienen sus implementaciones similares a PL/SQL:
  - T-SQL (Transact-SQL): Extensión de Microsoft (SQL Server) y Sybase.
  - PL/PgSQL (Procedural Language/PostgreSQL Structured Query Language): Lenguaje en PostgreSQL.
  - MySQL no tiene ningún lenguaje específico, pero incluye muchas de las características de estos otros lenguajes para la creación de procedimientos y funciones almacenadas.
- **Los códigos desarrollados en estos lenguajes se pueden almacenar como objetos de la base de datos, creando funciones y procedimientos y reutilizándose por otros usuarios.**
- Los disparadores o triggers se programan utilizando estos lenguajes.

BLOQUES DE CÓDIGO ANÓNIMOS

- El código más básico en PL/SQL son los bloques anónimos que se pueden introducir y ejecutar directamente en la consola SQL.
- Cada SGBD indica la forma de definir estos bloques.
- MySQL no permite el uso de bloques anónimos, por lo que crearemos nuestros bloques inicialmente como procedimientos (más adelante se explican como funcionan).
- La estructura de un bloque anónimo en PL/SQL ES:

```
[DECLARE]
-- variables, cursores, excepciones definidas por el usuario
BEGIN
   -- Sentencias SQL
   -- Sentencias de control PL/SQL
[EXCEPTION]
   -- Acciones cuando se producen errores
END;
```

ESCRIBIR CÓDIGO EN PROCEDIMIENTOS ALMACENADOS

- Crear un procedimiento:

```
DELIMITER //
CREATE PROCEDURE nombre ()
BEGIN
        SENTENCIAS
END //
DELIMITER ;
```

- Ejecutar un procedimiento:

```
CALL nombre ();
```

CONTENIDO DE LOS BLOQUES DE CÓDIGO

- El bloque de código estará formado por dos partes:
  - Declaración de variables
- Conjunto de sentencias:
  - Secuencias
  - Alternativas, selectivas o condicionales
  - Iteraciones, bucles o repeticiones

DECLARACIÓN Y ASIGNACIÓN DE VALORES A VARIABLES

- Declaración:

```
DECLARE nombreDeVariable tipo;
```

```
DECLARE miPrimeraVariable INT;
DECLARE saludo VARCHAR(50);
DECLARE miTerceraVariable INT DEFAULT 15;
```

- Asignación:

```
SET nombreDeVariable = valor;
```

```
SET miPrimeraVariable = 7;
SET miPrimeraVariable = 5 * 3;
SET miPrimeraVariable = miPrimeraVariable * 2;
SET saludo = 'Buenas tardes';
SET saludo = concat (saludo, ' Antonio');
```

CREACIÓN Y ÁMBITO DE VARIABLES

```
DELIMITER //
CREATE PROCEDURE ambito2 ()
BEGIN
  DECLARE x1 CHAR(25) DEFAULT 'soy la variable de fuera';
  BEGIN
     DECLARE x1 CHAR(50) DEFAULT 'soy la variable de dentro';
     SELECT x1;
     SET x1 ='Esta asignacion en realidad no sirve para nada';
  END;
  SELECT x1;
END; //
DELIMITER ;
```

El ALCANCE de las variables es únicamente el bloque BEGIN – END en el que se han declarado, es decir:
Las variables sólo "están vivas" dentro del bloque BEGIN-END en el que se han declarado.

VARIABLES EN SCRIPTS DE MYSQL

- MySQL permite utilizar variables para almacenar ciertos resultados. Estas variables pueden utilizarse después. Las variables se identifican porque van precedidas por un carácter '@'.

```
mysql> select count(*) into @var from mysql.user;
Query OK, 1 row affected (0.00 sec)

mysql> select @var;
+------+
| @var |
+------+
|    6 |
+------+
1 row in set (0.00 sec)
```

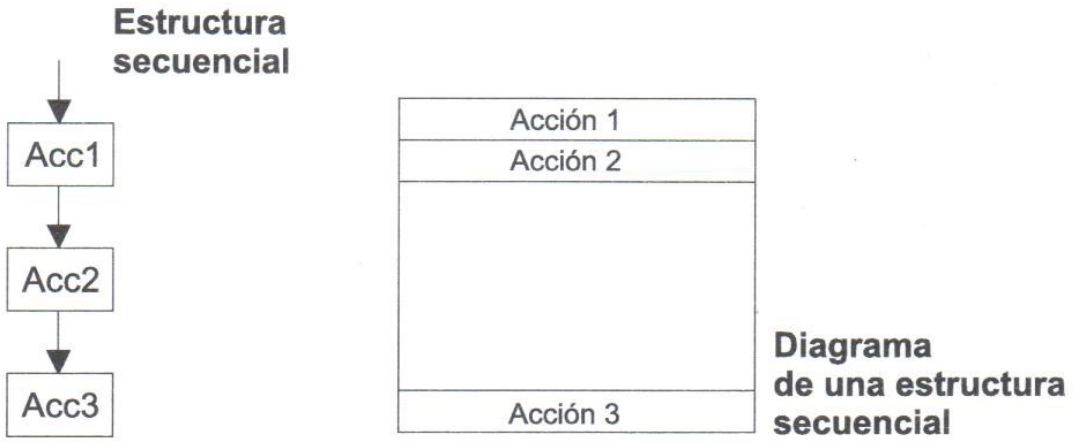
Se puede asignar un valor a una variable con el comando "SET".

```
mysql> set @var=10;
```

DISEÑO ESTRUCTURADO: ESTRUCTURAS BÁSICAS DE CONTROL

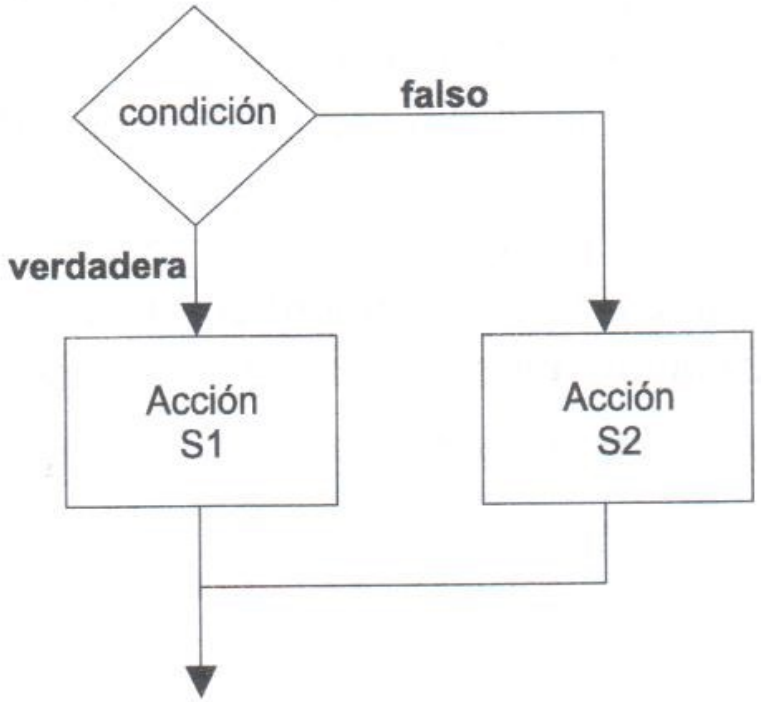
En 1966, Bhöm y Jacopini demostraron que un programa puede ser escrito utilizando solamente tres tipos de estructuras de control: secuenciales, selectivas y repetitivas:

- En la estructura **secuencial**, una acción o instrucción sigue a otra en secuencia y las tareas se suceden de tal modo que la salida de una es la entrada de la siguiente y así sucesivamente hasta el fin del proceso.



DISEÑO ESTRUCTURADO: ESTRUCTURAS BÁSICAS DE CONTROL

- En las estructuras **selectivas** o **condicionales** se evalúa una condición, utilizando expresiones lógicas, y en función del resultado de la misma se realiza una opción u otra. Si se cumple la condición, se realiza la tarea de la Parte SI. Si no se cumple, se realiza la tarea de la parte NO.



Pseudocódigo

Sí < condición >
  Entonces < acción S1 >
  Sino < acción S2 >
Fin_si

Obsérvese que en el pseudocódigo las palabras reservadas *entonces* y *sino* están indentadas en relación con las palabras *si* y *fin_si*; este procedimiento aumenta la legibilidad de la estructura y es el medio más idóneo para representar algoritmos.

ESTRUCTURAS DE PROGRAMACIÓN: IF – THEN - ELSE

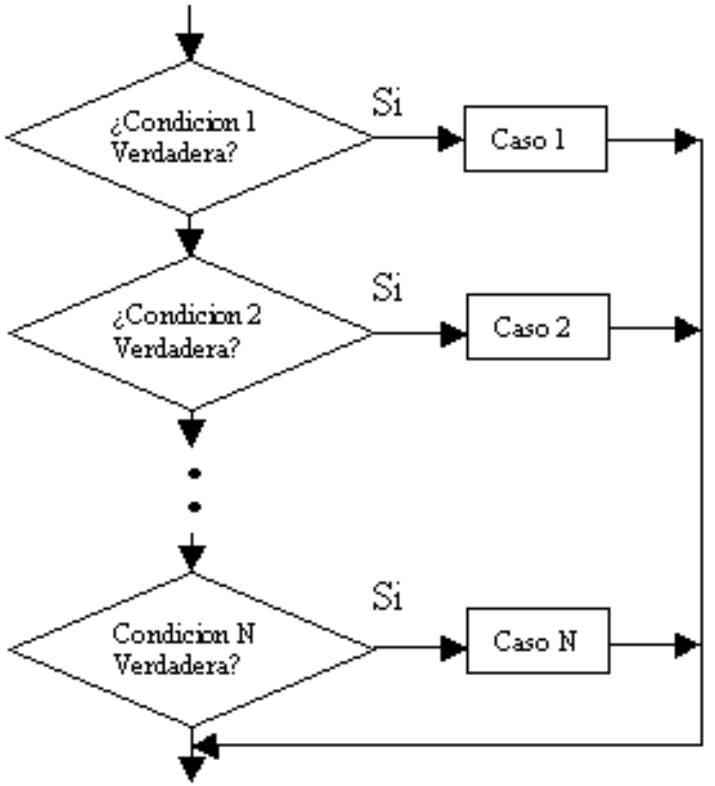
```
IF expresión1 THEN
    xxxxxxx
    xxxxxxx
ELSEIF expresión2 THEN
    xxxxxxx
    xxxxxxx
ELSE
    xxxxxxx
    xxxxxxx
END IF;
```

```
DELIMITER $$
CREATE PROCEDURE proc7 (IN par1 INT)
BEGIN
DECLARE var1 INT;
SET var1 = par1 + 1;
IF var1 = 0 THEN
INSERT INTO t VALUES (17);
END IF;
IF par1 = 0 THEN
UPDATE t SET s1 = s1 + 1;
ELSE
UPDATE t SET s1 = s1 + 2;
END IF;
END; $$
```

DISEÑO ESTRUCTURADO: ESTRUCTURAS BÁSICAS DE CONTROL

Selección múltiple

```
En-caso-de <expresión> haga
  Caso <opción 1>:
        <instrucciones>
  caso <opción 2>:
        <instrucciones>
  caso <opción 3>:
        <instrucciones>
...
  caso <opción N>:
       <instrucciones>
SINO <instrucciones a realizar si no
       se ha cumplido Ninguna de
       las condiciones anteriores>
Fin-Caso
```



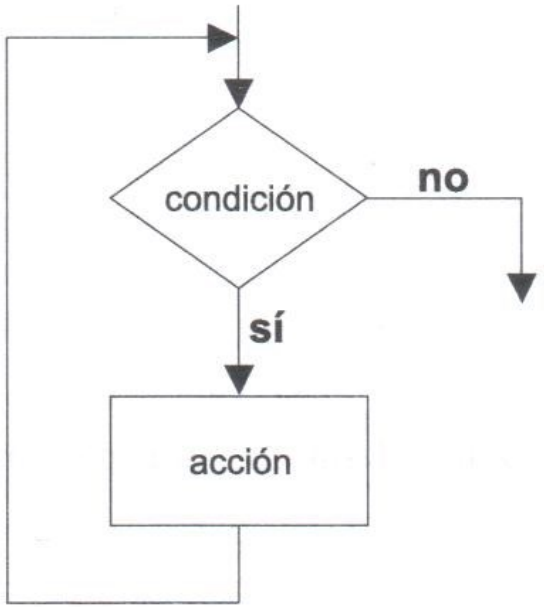
ESTRUCTURAS DE PROGRAMACIÓN: CASE

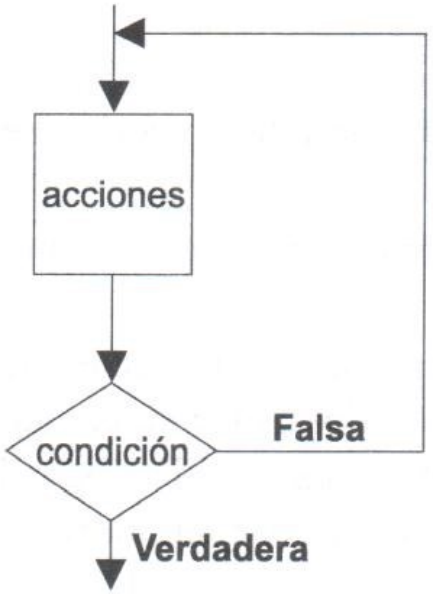
```
CASE expresión1
    WHEN valor1 THEN
        xxxxxxx
        xxxxxxx
    WHEN valor2 THEN
        xxxxxxx
        xxxxxxx
    WHEN valor3 THEN
        xxxxxxx
        xxxxxxx
    ELSE
        xxxxxxx
        xxxxxxx
END CASE;
```

```
CREATE PROCEDURE proc8(IN parameter1 INT)
BEGIN
DECLARE variable1 INT;
SET variable1 = parameter1 + 1;
CASE variable1
WHEN 0 THEN INSERT INTO t VALUES (17);
WHEN 1 THEN INSERT INTO t VALUES (18);
ELSE INSERT INTO t VALUES (19);
END CASE;
END; $$
```

DISEÑO ESTRUCTURADO: ESTRUCTURAS BÁSICAS DE CONTROL

- Las estructuras **repetitivas** que repiten un número determinado de veces una secuencia de instrucciones, se denominan bucles, llamándose interacción al hecho de repetir la ejecución de una secuencia de acciones. Existen dos tipos de construcciones repetitivas: Hacer mientras y repetir hasta.





Pseudocodigo

Repetir

< acciones >

.

.

hasta_que < condición >

El bucle *repetir-hasta_que* se repite mientras el valor de la expresión booleana de la condición sea falsa, justo la opuesta de la sentencia mientras.

ESTRUCTURAS DE PROGRAMACIÓN: WHILE DO

Comprueba la expresión antes de ejecutar las instrucciones.

[Etiqueta:] WHILE expresión DO

xxxxx

xxxxx

END WHILE [Etiqueta];

```
DELIMITER $$
CREATE PROCEDURE proc10()
DECLARE i int;
SET i=1;
loop1: WHILE i<=10 DO
          IF MOD(i,2)<>0 THEN
               SELECT CONCAT(i," es impar");
          END IF;
          SET i=i+1;
END WHILE loop1;
```

ESTRUCTURAS DE PROGRAMACIÓN: REPEAT UNTIL

Comprueba la expresión **después** de ejecutar las instrucciones.

[Etiqueta:] REPEAT

    xxxxx

    xxxxx

UNTIL expresión

END REPEAT;

END [Etiqueta];

```
DELIMITER $$
CREATE PROCEDURE proc10()
BEGIN
DECLARE i int;
SET i=0;
loop1: REPEAT
          SET i=i+1;
IF MOD(i,2)<>0 THEN /*número impar*/
select concat(i," es impar");
END IF;
UNTIL i >= 10
END REPEAT;
END; $$
```

ESTRUCTURAS DE PROGRAMACIÓN: LOOP

```
[Etiqueta:] LOOP
    xxxxx
    Xxxxx
    [LEAVE etiqueta;]
END LOOP
[Etiqueta:];
```

```
DELIMITER $$
CREATE PROCEDURE proc9()
BEGIN
      DECLARE cont INT;
      SET cont = 0;
      loop_label: LOOP
            INSERT INTO t VALUES (cont);
            SET cont = cont + 1;
            IF cont >= 5 THEN
                  LEAVE loop_label;
            END IF;
      END LOOP;
END; $$
```

SINTAXIS DE PROCEDIMIENTO ALMACENADO

```
DELIMITER //
    CREATE PROCEDURE miProcedimiento (IN id INT, ...)
    BEGIN
        SELECT * FROM miTabla WHERE miCampo=id;
    END //
DELIMITER ;  ← Espacio entre "DELIMITER" y ";"
```

```
CALL miProcedimiento (parámetro);
SOURCE miArchivoDeProcedimientos.sql;
SHOW CREATE PROCEDURE miProcedimiento;

DROP PROCEDURE miProcedimiento;
```

→ Los parámetros pueden ser IN, OUT o INOUT

REGLAS PARA NOMBRES DE SCRIPTS

- Los nombres de scripts no distinguen entre mayúsculas y minúsculas.

- No puede haber dos scripts con el mismo nombre en la misma base de datos de MySQL.

SINTAXIS DE FUNCIÓN ALMACENADA

```
DELIMITER //
    CREATE FUNCTION miFuncion (id INT) RETURNS FLOAT
    BEGIN
        DECLARE resultado FLOAT DEFAULT 0;
        SET resultado=id*2;
        RETURN resultado;
    END //
DELIMITER ;  ← Espacio entre "DELIMITER" y ";"
```

```
SELECT miFuncion(3);
SOURCE miArchivoDeFunciones.sql;
SHOW CREATE FUNCTION miFuncion;

DROP FUNCTION miFuncion;
```

→ Los parámetros sólo pueden ser IN

SCRIPTS ALMACENADOS

Procedimientos almacenados	Funciones almacenadas
CREATE PROCEDURE	CREATE FUNCTION
Se invocan con el comando CALL	Se pueden utilizar en expresiones así como en el interior de sentencias SQL
Tienen parámetros IN, OUT e INOUT	Sólo tienen parámetros IN
Pueden devolver varios valores a través de sus parámetros OUT e INOUT	Devuelven valores a través de la instrucción RETURN
Pueden llamar a otras rutinas almacenadas	Pueden llamar a otras rutinas almacenadas
Están asociados a una base de datos concreta	Están asociados a una base de datos concreta

INSTRUCCIONES NO PERMITIDAS DENTRO DE PROCEDIMIENTOS Y FUNCIONES

- CREATE PROCEDURE
- ALTER PROCEDURE
- DROP PROCEDURE
- CREATE FUNCTION
- DROP FUNCTION
- CREATE TRIGGER
- DROP TRIGGER

FUNCIONES ÚTILES PARA LOS SCRIPTS SQL

Función	Utilidad
row_count()	Devuelve el número de filas que se actualizaron tras ejecutar una operación de actualización.
last_insert_id()	Devuelve el identificador autonumérico de la última fila insertada.
user()	Devuelve el nombre del usuario conectado
database()	Devuelve el nombre de la base de datos que se está usando
current_user()	Devuelve el nombre de usuario y host desde el que se conecta
current_date()	Devuelve la fecha actual
current_time()	Devuelve la hora actual
version()	Devuelve la versión del servidor
found_rows()	Devuelve el número de filas de la última select ejecutada
rand()	Devuelve un número aleatorio
sleep(segundos)	Espera un número determinado de segundos.

ALGUNAS FUNCIONES CON CADENAS

Función	Ejemplo
CHAR_LENGTH Cálculo de longitud de cadena en caracteres.	SELECT char_length('Hola') --> 4
CONCAT Concatena dos cadenas de caracteres.	SELECT concat ('Ho', 'l', 'a') --> Hola
INSERT Inserta una cadena en otra.	`SELECT insert('Edificio', 3, 4, 'What'); --> EdWhatio`
INSTR Busca una cadena en otra.	SELECT instr('Edificio', 'ifi');-->3
REPLACE Busca una secuencia en una cadena y la sustituye por otra.	SELECT replace('Edificio', 'ifi', 'ofo'); --> Edofocio
REVERSE Invierte el orden de los caracteres de una cadena.	SELECT reverse('12345'); --> '54321'
STRCMP Compara cadenas.	SELECT STRCMP('text', 'text2'); → -1 SELECT STRCMP('text2', 'text'); → 1 SELECT STRCMP('text', 'text'); → 0

ALGUNAS FUNCIONES MATEMÁTICAS

Función	Ejemplo
ABS Devuelve el valor absoluto	SELECT abs(-15); → 15
MOD Resto de una división entera	SELECT mod(554, 10) → 4
RAND Valores aleatorios	SELECT rand(); → 0.80785042161585
ROUND Cálculo de redondeos	SELECT round(-1.78); → -2
SIGN Devuelve el signo	SELECT sign(-15); → -1
SQRT Cálculo de la raíz cuadrada	SELECT sqrt(4); → 2
TRUNCATE Elimina decimales	SELECT truncate (1.723,1); →1.7

ALGUNAS FUNCIONES DE FECHA Y HORA

Función	Ejemplo
current_date() Fecha actual	SELECT current_date(); → 2017-12-17
current_time() Hora actual	SELECT current_time() → 10:40:07
now() Fecha y hora actuales	SELECT now(); → 2017-12-17 10:40:08
dateDiff(unaFecha, otraFecha) Días entre dos fechas	SELECT dateDiff('2017-12-17', '2017-12-20'); → - 3
day(unaFecha); El día de una fecha dayName(unaFecha); El nombre del día de una fecha	SELECT day('2017-12-17'); →27 SELECT dayName('2017-12-17'); → Wednesday
month(unaFecha); El mes de una fecha monthName(unaFecha); El nombre del mes de una fecha	SELECT month('2017-12-17'); → 12 SELECT monthName('2017-12-17'); → December
year(unaFecha) ; El año de una fecha	SELECT year('2017-12-17'); → 2017

Las fechas por defecto van en formato AAAA-MM-DD y en inglés

Cambiar de formato → date_format(unaFecha, "%d-%m-%Y")
SELECT date_format('2017-12-17', "%d-%m-%Y") → 17-12-2017

Cambiar de idioma para fechas → SET lc_time_names=es_ES

GESTIÓN DE ERRORES (EXCEPCIONES)

- Tenemos dos tipos de manejadores de error:
  - EXIT (en ese punto **termina** la ejecución del bloque begin end)
  - CONTINUE (la ejecución del procedimiento **continúa** una vez hayamos tratado el error)
- Tratar el error = ejecutar lo que hayamos determinado.
- Sintaxis general de los manejadores de error:

```
DECLARE { EXIT ó CONTINUE }
HANDLER FOR  númeroDeError  /  SQLSTATE nombreDeError / condición
instrucciones (Código a ejecutar en el caso de que se produzca el error)
```

- Existen 3 condiciones predefinidas:

  NOT FOUND (no hay más filas)
  SQLEXCEPTION (error)
  SQLWARNING (warning)

EJEMPLO DE MANEJADOR DE ERROR DE TIPO EXIT I

```
CREATE TABLE t2
(
  s1 INT,
  PRIMARY KEY (s1)
);
```

```
CREATE TABLE t3
(
  s1 INT,
  FOREIGN KEY (s1) REFERENCES t2 (s1)
);
```

INSERT INTO t3 VALUES (5); ← *al ejecutar esto se produce un error*

```
mysql> INSERT INTO t3 VALUES (5);
ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint fails
(`labd9`.`t3`, CONSTRAINT `t3_ibfk_1` FOREIGN KEY (`s1`) REFERENCES `t2` (`s1`))
mysql>
```

EJEMPLO DE MANEJADOR DE ERROR DE TIPO EXIT II

```
CREATE TABLE listadoErrores
(
    mensajeError CHAR(80)
);
```

```
DELIMITER //
```

```
CREATE PROCEDURE insertConManejadorExit(IN parametro INT)
BEGIN
  DECLARE EXIT HANDLER FOR 1452
      INSERT INTO listadoErrores VALUES  (CONCAT('Time: ', current_date, '. Fallo de FK para
      parámetro= ', parametro));
  INSERT INTO t3 VALUES (parametro);
END; //
```

```
DELIMITER ;
```

COMPARAMOS LA SINTAXIS GENERAL CON EL MANEJADOR DEL EJEMPLO TIPO EXIT

DECLARE { EXIT ó CONTINUE }

HANDLER FOR número_de_error / SQLSTATE nombre_de_error / condición

instrucciones (*Código a ejecutar en el caso de que se produzca el error*)

```
DECLARE EXIT HANDLER FOR 1452
   INSERT INTO listadoErrores VALUES  (CONCAT('Time: ', current_date, '. Fallo de FK
    para parámetro= ', parametro));
```

Veámoslo en detalle...

DECLARE EXIT ← Manejador de tipo EXIT (una vez se produce un error no se ejecuta nada más en el procedimiento)

HANDLER FOR 1452 ← Hemos elegido identificar el error por número_de_error

INSERT INTO listadoErrores VALUES (CONCAT('Time: ', current_date, '. Fallo de FK para parámetro= ', parametro)); ← instrucciones (en este caso sólo hay una instrucción que consiste en escribir en una tabla)

EJEMPLO DE MANEJADOR DE ERROR DE TIPO CONTINUE

```
CREATE TABLE t4
(
    s1 INT,
    primary key(s1)
);
```

```
DELIMITER //
```

```
CREATE PROCEDURE ejemploManejadorContinue ()
BEGIN
        DECLARE CONTINUE HANDLER  FOR  SQLSTATE '23000' SET @x2 = 1;
        SET @x = 1;
        INSERT INTO t4 VALUES (1);
        SET @x = 2;
        INSERT INTO t4 VALUES (1);
        SET @x = 3;
END; //
```

```
DELIMITER ;
```

COMPARAMOS LA SINTAXIS GENERAL CON EL MANEJADOR DEL EJEMPLO TIPO CONTINUE

DECLARE { EXIT ó CONTINUE }

HANDLER FOR número_de_error / SQLSTATE nombre_de_error / condición

instrucciones (Código a ejecutar en el caso de que se produzca el error)

DECLARE CONTINUE HANDLER FOR SQLSTATE '23000' SET @x2 = 1;

Veámoslo en detalle...

DECLARE CONTINUE ← Manejador de tipo CONTINUE

HANDLER FOR SQLSTATE '23000' ← Hemos elegido identificar el error por SQLSTATE nombre_de_error

SET @x2 = 1; ← instrucciones (en este caso sólo hay una instrucción)

CURSORES

- Dentro de las funciones y procedimientos podemos trabajar con instrucciones que pertenecen al DML.
- Cuando dichas instrucciones involucran más de una fila (varios registros) necesitamos utilizar una variable especial para almacenar la información. Esta variable se llama CURSOR.
- Con el cursor podemos recorrer la información registro por registro.

CURSORES

¿Qué podemos hacer con un cursor?

→ DECLARE nombreDelCursor CURSOR FOR SELECT ...;

→ OPEN nombreDelCursor;

→ FETCH nombreDelCursor INTO unaVariable [, otraVariable];

→ CLOSE nombreDelCursor;

FETCH = Extraer datos del cursor

Los cursores son de sólo-lectura, no se puede hacer UPDATE de un cursor. Se leen línea a línea y sólo en un sentido (siempre para delante, no se puede volver a una fila que ya se ha leído)

Ejemplo:

```
DECLARE miCursor CURSOR FOR SELECT campo1, campo2,
   campo3 FROM unaTabla;
```

CURSORES

```
DECLARE miCursor CURSOR FOR SELECT campo1, campo2, campo3 FROM unaTabla;
```

```
DECLARE cursorEjemplo CURSOR FOR SELECT first_name, last_name FROM actor;
```

CURSORES

```
DROP PROCEDURE IF EXISTS procCursor;
DELIMITER //
CREATE PROCEDURE procCursor (codigoActor INT)
BEGIN
  DECLARE apellido VARCHAR(45);
  DECLARE miCursor CURSOR FOR
  SELECT first_name FROM actor WHERE actor_id=codigoActor;
  OPEN miCursor;
  LOOP
      FETCH miCursor INTO apellido;
   END LOOP;
   CLOSE miCursor;
END; //
DELIMITER ;
```

```
mysql> call procCursor(5);
ERROR 1329 (02000): No data - zero rows fetched, selected, or processed
mysql>
```

CURSORES CON EXCEPCIONES

```
DROP PROCEDURE IF EXISTS procCursor;
DELIMITER //
CREATE PROCEDURE procCursor (IN codigoActor INT,  OUT apellido VARCHAR(45), OUT
    nombre varchar(45))
BEGIN
  DECLARE fin INT DEFAULT 0;
  DECLARE miCursor CURSOR FOR
  SELECT first_name , last_name FROM actor WHERE actor_id=codigoActor;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET fin = 1;
  OPEN miCursor;
  REPEAT
      FETCH miCursor INTO apellido, nombre;
   UNTIL fin = 1   END REPEAT;
   CLOSE miCursor;
END; //
DELIMITER ;
```

ORDEN DE LAS DECLARACIONES

- Declarar variables
- Declarar cursores
- Declarar manejadores

```
DECLARE a,b INT;
DECLARE cur_1 CURSOR FOR SELECT s1 FROM t;
DECLARE CONTINUE HANDLER FOR NOT FOUND
```

UT10. SQL MODO PROGRAMACIÓN

Módulo: BASES DE DATOS

Curso 2022/2023. 1° DAM

Ruth Lospitao Ruiz