BBDD

Bases de Datos – Programación de bases de datos II

IES Ciudad Escolar
Desarrollo de Aplicaciones Multiplataforma

Manejo de errores

- Cuando se desencadena un error en MySQL bien sea en el servidor o en el cliente, este generalmente incluye la siguiente información: Código de error, valor de SQLSTATE y un mensaje asociado:

– **Error code**: valor numérico específico de MySQL (distinto por tanto de otros SGBDs) que identifica el error en cuestión. Se categorizan por rangos (Ej1: entre 1000 y 1999 son errores del servidor enviados al cliente. Ej2: entre 2,000 y 2,999 son errores de cliente.)

– **SQLSTATE**: 5 caracteres, estándar de SQL para los errores. Los dos primeros caracteres identifican la naturaleza del error: “00” (SUCCESS), “01” (SQLWARNING), “02” (NOT FOUND para cursores) o bien > “02” (SQLEXCEPTION).

– **Message string**: Cadena de texto que nos da una descripción del error en cuestión.

https://dev.mysql.com/doc/refman/8.0/en/error-message-elements.html#error-elements

- Ya sabemos disparar nuestros propios errores y excepciones con la sentencia **SIGNAL.** Con ella podemos personalizar el *error code, sqlstate y message text* que queramos.

```
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = ‘xxxxxxxxxx’;
```

Manejo de errores

- Al igual que sucede en JAVA, nosotros podemos capturar excepciones y ejecutar el código adicional que consideremos en caso de dispararse una excepción.
- Para capturar una excepción en MySQL utilizamos **handlers** (manipuladores) que serían equivalentes al nuestros **try – catch** de JAVA.
- El código que usualmente ponemos en el **catch { }** de JAVA, en MySQL lo debemos indicar en el cuerpo del handler (entre el begin y end).
- Las opciones del *handler_action* son:
  - **CONTINUE**: La ejecución continúa.
  - **EXIT**: La ejecución termina.
  - **UNDO**: No está soportado en MySQL.
- MySQL permite capturar **Error code y SQLSTATE** concretos o situaciones genéricas como **NOT FOUND, SQLWARNING** o **SQLEXCEPTION**

```
DECLARE handler_action HANDLER
    FOR condition_value [, condition_value] ...
    statement

handler_action: {
    CONTINUE
  | EXIT
  | UNDO
}

condition_value: {
    mysql_error_code
  | SQLSTATE [VALUE] sqlstate_value
  | condition_name
  | SQLWARNING
  | NOT FOUND
  | SQLEXCEPTION
}
```

https://dev.mysql.com/doc/refman/8.3/en/declare-handler.html

```
DECLARE EXIT HANDLER FOR SQLEXCEPTION
 BEGIN
  -- Código a ejecutar si se dispara una excepción
 END;
```

```
DECLARE CONTINUE HANDLER FOR NOT FOUND
 BEGIN
  -- Código a ejecutar si se dispara una excepción
 END;
```

Manejo de errores

- Los HANDLERS deben declararse inmediatamente después de las variables locales.
- Si capturamos un error o excepción con un handler, ya no nos saldrá por consola el motivo de ambos.
- Para ver el detalle del error concreto podemos añadir al cuerpo del HANDLER (independientemente del código que ya fuéramos a realizar en caso de dispararse):

```
GET DIAGNOSTICS CONDITION 1
@p1 = RETURNED_SQLSTATE, @p2 = MESSAGE_TEXT;
SELECT @p1 as RETURNED_SQLSTATE, @p2 as MESSAGE_TEXT;
```

- Cuando ejecutamos una sentencia SQL en el cliente de MySQL podemos ver por defecto en consola únicamente el error y mensaje de texto asociado. Si queremos ver también el SQLSTATE, podríamos ejecutar lo siguiente tras cada sentencia a evaluar:

```
GET DIAGNOSTICS CONDITION 1
@p1 = RETURNED_SQLSTATE, @p2 = MESSAGE_TEXT,  @p3 = MYSQL_ERRNO;
SELECT @p1 as RETURNED_SQLSTATE, @p2 as MESSAGE_TEXT, @p3 as MYSQL_ERRNO;
```

- Al igual que en JAVA también podemos disparar una excepción dentro del código que gestiona otra excepción…

```
DECLARE EXIT HANDLER FOR SQLEXCEPTION
  BEGIN
    -- Código a ejecutar si se dispara una excepción
    SIGNAL sqlstate '45000' set message_text =  …
  END;
```

Manejo de errores: Transacciones

- Por defecto, si se produce un error interno en la alguna sentencia que forma parte de una transacción, NO se hace ROLLBACK automáticamente. Se completan todas las sentencias, unas con error y otras correctamente.

- Para realizar ROLLBACK automáticamente ante un error cualquiera en nuestro código almacenado, debemos capturar la excepción que se dispara mediante un **exit *handler*** para que deshaga los posibles cambios realizados hasta entonces. En caso de no dispararse, se completará la transacción satisfactoriamente y se dará persistencia a todos los cambios mediante COMMIT.

- Por ejemplo:

```
DELIMITER $$
CREATE PROCEDURE procedimiento ()
BEGIN

  DECLARE EXIT HANDLER FOR SQLEXCEPTION
  BEGIN
            ROLLBACK;
  END;

  START TRANSACTION;
            SENTENCIA  SQL 1;
            SENTENCIA  SQL 2;
            SENTENCIA  SQL …;
  COMMIT;

END $$
DELIMITER ;
```

De esta forma, si la sentencia SQL 1, 2 o … de la transacción no se ejecuta satisfactoriamente, con el primer error que surja, se dispararía una excepción que al capturarla, nos permitiría automatizar el **rollback**, deshaciendo así cualquier cambio previo dentro de la transacción.

En caso de que ninguna sentencia SQL dentro de la transacción fallara, finalmente se realizaría el **commit**.