Manejo de errores.

1 Introducción a la gestión de errores.

En general, si una sentencia SQL falla dentro de un programa almacenado se produce una situación de error, se interrumpe la ejecución del programa en ese punto y finaliza salvo en el caso de que el programa que falla hubiera sido llamado por otro; en ese caso la ejecución continua por el programa que llamó a este programa que ha causado el error; ocurriría lo mismo si en lugar de un programa que es llamado desde otro programa nos encontráramos con un bloque interno dentro de otro bloque más externo; si se produjera error en el bloque anidado interno, la ejecución del programa se interrumpiría en el bloque interno para continuar por el externo.
Este comportamiento se puede controlar definiendo los **manejadores de error o handlers.**
Una handler es un bloque de instrucciones SQL que se ejecuta cuando se verifica una condición tras una excepción (error) generada por el servidor.

Sintaxis:

DECLARE {CONTINUE | EXIT | UNDO} HANDLER FOR
{SQLSTATE sqlstate_code | MySQL error code | nombre_condición} instrucciones_del_manejador

Deben declararse después de las declaraciones de variables y cursores ya que referencian a estos en su declaración.

El manejador puede ser de tres tipos:

• **CONTINUE**: La excepción o error generado no interrumpe el código del procedimiento.
• **EXIT**: Permite poner fin al bloque de instrucciones o programa en el que se genera la excepción.
• **UNDO**: No está soportada por el momento.

La condición de error del manejador puede expresarse también de tres formas:

• Con un código estándar ANSI SQLSTATE.
• Con un código de error MySQL.
• Una expresión.

El manejador de error indica mediante un conjunto de instrucciones lo que hay que hacer en caso de que se produzca ese error.

Ejemplos de situaciones de error:

```
2 DELIMITER $$
3 DROP PROCEDURE IF EXISTS error1 $$
4 CREATE PROCEDURE error1 (p_id INT, p_alumno VARCHAR(30))
5 MODIFIES SQL DATA
6 BEGIN
7     INSERT INTO alumnos VALUES(p_id,p_alumno);
8 END $$
```

El anterior procedimiento recibe dos argumentos, identificador de alumno y nombre del alumno y los inserta en la base de datos. Como modifica la base de datos se intercala la cláusula de la línea 5.

Si hacemos una llamada al anterior procedimiento desde la ventana cliente o desde la propia herramienta MySQL Query Browser:

```
CALL error1(6, 'Alberto Carrera');
```

Qué ocurre si intentamos insertar un alumno de clave primaria repetida?

CALL error1(6, 'Raquel M. Carrera');

Aparecerá un error con un número de error (1062) y una descripción (“entrada duplicada”) advirtiéndonos de tal situación; evidentemente no se realiza la inserción y la ejecución del programa se detiene y finaliza (pues no retorna a ningún otro programa ya que no hay ningún otro que lo llamó).
Tomando nota del número de error y modificando el anterior procedimiento para añadir dicho control de error:

```
2  DELIMITER $$
3  DROP PROCEDURE IF EXISTS error2 $$
4  CREATE PROCEDURE error2 (p_id INT, p_alumno VARCHAR(30))
5  MODIFIES SQL DATA
6  BEGIN
7      DECLARE CONTINUE HANDLER FOR 1062
8        SELECT CONCAT('Nº de matrícula ', p_id, ' ya existente') AS 'Aviso de error';
9      INSERT INTO alumnos VALUES(p_id,p_alumno);
10 END $$
```

Una llamada al programa como:

CALL error2(6, 'Raquel M. Carrera');

Producirá la siguiente salida:

Aviso de error
Nº de matrícula 6 ya existente

Resaltar que el error ha sido tratado y por tanto no finaliza la ejecución del programa en el momento en que se produce y si hubiera más líneas de código detrás de la línea 9, éstas se ejecutarían debido al tipo de manejador, CONTINUE; esto no ocurriría si el error no fuera tratado como hemos comprobado en el procedimiento error1 anterior.

¿Qué ocurre si intentamos introducir un alumno de clave primaria nula?

CALL error2(NULL, 'Mario A. Carrera');

Habrá que considerar y tratar también el error 1048 que se produce:

```
2  DELIMITER $$
3  DROP PROCEDURE IF EXISTS error3 $$
4  CREATE PROCEDURE error3 (p_id INT, p_alumno VARCHAR(30))
5  MODIFIES SQL DATA
6  BEGIN
7      DECLARE CONTINUE HANDLER FOR 1062
8        SELECT CONCAT('Nº de matrícula ', p_id, ' ya existente') AS 'Aviso de error';
9      DECLARE CONTINUE HANDLER FOR 1048
10       SELECT CONCAT('El nº de matrícula no puede ser nulo') AS 'Aviso de error';
11     INSERT INTO alumnos VALUES(p_id,p_alumno);
12 END $$
```

Llegados a este punto podemos utilizar un tercer argumento en el procedimiento, **un parámetro de tipo OUT**, para que actúe como flag o indicador de cómo ha ido la ejecución del procedimiento:

```
2  DELIMITER $$
3  DROP PROCEDURE IF EXISTS error4 $$
4  CREATE PROCEDURE error4 (p_id INT, p_alumno VARCHAR(30), OUT p_resultado VARCHAR(40))
5  MODIFIES SQL DATA
6  BEGIN
7      DECLARE CONTINUE HANDLER FOR 1062
8        SET p_resultado = CONCAT('Nº de matrícula ', p_id, ' ya existente');
9      DECLARE CONTINUE HANDLER FOR 1048
10       SET p_resultado = 'Nº de matrícula nulo';
11     SET p_resultado = 'correcto';
12     INSERT INTO alumnos VALUES(p_id,p_alumno);
13 END $$
```

Llamando al anterior procedimiento error4 desde el interior de otro (línea 7 de error5):

```
2  DELIMITER $$
3  DROP PROCEDURE IF EXISTS error5 $$
4  CREATE PROCEDURE error5 ()
5  BEGIN
6     DECLARE v_resultado VARCHAR(40);
7     CALL error4 (7, 'Mariano Carrera', v_resultado);
8     IF v_resultado = 'correcto' THEN
9       SELECT 'Alumno dado de alta' AS 'Insercion de alumnos';
10    else
11      SELECT v_resultado AS 'Aviso de error';
12    END IF;
13 END $$
```

La primera ejecución del procedimiento: CALL error5( ) funcionará correctamente, apareciendo como fila/columna resultante : ‘Alumno dado de alta’.
La segunda ejecución del procedimiento anterior también funcionará correctamente pues la situación de error de código repetido es tratada y por tanto el programa finaliza correctamente y visualiza por pantalla que el número de matrícula ya existe.

Volviendo otra vez a la declaración de un manejador, hemos visto que tiene 3 partes que pasaremos a ver a continuación más en detalle:

1. Tipo de manejador: CONTINUE o EXIT.
2. Condición del manejador: SQLSTATE sqlstate_code | MySQL error code | nombre_condición.
3. Acciones o instrucciones que realiza el manejador.

2 Tipos de manejador

• **EXIT**: El programa que causa el error finaliza, devolviendo el control al programa que le llamó. Si se produce en un bloque interno, entonces el control de la ejecución pasa al externo.
• **CONTINUE**: La ejecución continúa por la siguiente línea de código que causó el error.
En ambos casos el error es tratado por lo que la ejecución del programa puede considerarse correcta y antes de realizarse la opción CONTINUE o EXIT se ejecuta el conjunto de instrucciones asociados al manejador.

Ejemplo de utilización del manejador EXIT:

```
2  DELIMITER $$
3  DROP PROCEDURE IF EXISTS error6 $$
4  CREATE PROCEDURE error6 (p_id INT, p_alumno VARCHAR(30))
5  MODIFIES SQL DATA
6  BEGIN
7      DECLARE v_clave_repetida TINYINT DEFAULT 0;
8      BEGIN
9         DECLARE EXIT HANDLER FOR 1062
10          SET v_clave_repetida=1;
11        INSERT INTO alumnos VALUES(p_id,p_alumno);
12        SELECT CONCAT('Alumno dado de alta: ', p_alumno, ' con matrícula nº: ', p_id )
13              AS 'Inserción de alumnos';
14     END;
15     IF v_clave_repetida=1 THEN
16        SELECT CONCAT('Nº de matrícula: ', p_id,' ya existente') as 'Aviso de error';
17     END IF;
18 END $$
```

Observa como en este caso hay dos bloques de instrucciones BEGIN…END. El manejador de errores con la opción EXIT está integrado con el bloque más interno.
La llamada al procedimiento anterior:

CALL error6(8, 'Fernando Carrera');

Dará como resultado la inserción del alumno y su mensaje correspondiente de alumno dado de alta.

CALL error6(8, 'Conchita Martín');

Dará como resultado un aviso indicando que el número de matrícula 8 ya existe. La instrucción de la línea 11 falla (no se lleva a cabo por tanto la inserción) puesto que no puede haber dos filas distintas con el mismo valor en el campo id (al ser clave primaria) provocando el error de clave repetida que hace que se ejecute la instrucción asociada al manejador (línea 10). Como el tipo de manejador es de tipo EXIT, se interrumpe la ejecución del programa y se sale del bloque interno (si sólo hubiera habido un bloque se hubiera salido del programa); por tanto la instrucción de la línea 12 no llega a ejecutarse y la ejecución del programa sigue por la línea 15.

Ejemplo de utilización del manejador CONTINUE.
Observa que a diferencia del anterior caso, en este procedimiento no hay más que un bloque:

```
2  DELIMITER $$
3  DROP PROCEDURE IF EXISTS error7 $$
4  CREATE PROCEDURE error7 (p_id INT, p_alumno VARCHAR(30))
5  MODIFIES SQL DATA
6  BEGIN
7      DECLARE v_clave_repetida TINYINT DEFAULT 0;
8      DECLARE CONTINUE HANDLER FOR 1062
9          SET v_clave_repetida=1;
10     INSERT INTO alumnos VALUES(p_id,p_alumno);
11     IF v_clave_repetida=1 THEN
12        SELECT CONCAT('Nº de matrícula: ', p_id,' ya existente') as 'Aviso de error';
13     ELSE
14     SELECT CONCAT('Alumno dado de alta: ', p_alumno, ' con matrícula nº: ', p_id )
15           AS 'Inserción de alumnos';
16     END IF;
17 END $$
```

CALL error7(9, 'Pablo Martínez') ;

Dará como resultado la inserción del alumno Pablo y su mensaje correspondiente de alumno dado de alta. Como no falla la instrucción la expresión de la línea 11 es falsa y se ejecutará la línea 14.

CALL error7(9, 'Luis Hueso') ;

Dará como resultado un aviso indicando que ese número de matrícula ya existe. Al intentar hacer la inserción de la línea 10, el procedimiento provoca una situación de error (igual que para el caso anterior de la cláusula EXIT) que es tratada inmediatamente después en el manejador de error poniendo a 1 el valor de la variable v_clave_repetida; por tanto, al ser tratado el error y de forma continua (cláusula CONTINUE línea 8) la ejecución del procedimiento continúa por la línea 11 haciendo cierta la condición y avisando de la duplicidad de dicha matrícula.

3 La condición del manejador.

Has 3 formas posibles de indicar cuando debe ser invocado el conjunto de instrucciones del manejador de error para tratarlo:

1) Número de error de MySQL.
2) Código SQLSTATE estándar ANSI.
3) Condiciones de error con nombre.

1) Número de error de MySQL, propio de MySQL, que dispone de un conjunto particular de números de error. Ej.:
DECLARE CONTINUE HANDLER FOR 1062 SET v_clave_repetida=1;
El número de error 1062 está asociado en MySQL al intento de almacenar un registro de clave duplicada.

2) Código SQLSTATE estándar ANSI: A diferencia del anterior, SQLSTATE no es definido por MySQL sino que es un estándar y, por tanto, el mismo error tiene asociado el mismo SQLSTATE independientemente del gestor de bases de datos utilizado: MySQL, Oracle, DB2, Microsoft SQL Server. En los anteriores gestores de bases de datos el SQLSTATE 23000 está asociado al error de clave duplicada. Por tanto en MySQL el anterior manejador de error y el siguiente son similares:

DECLARE CONTINUE HANDLER FOR SQLSTATE '23000' SET v_clave_repetida=1;

¿Cuál utilizar de los dos anteriores? En teoría el segundo permitiría una mayor portabilidad a otros gestores de bases de datos pero en la práctica los lenguajes de programación para Oracle y Microsoft SQL Server son incompatibles con el de MySQL por lo que no hace muy atractivo el uso de esta segunda opción (DB2 de IBM es “algo” compatible pues tanto DB2 como MySQL están basados en el estándar SQL:2003). Además hay códigos SQLSTATE que no corresponden con un código de error de MySQL sino a varios (para estos casos se utiliza el SQLSTATE genérico ‘HY000’).

Por lo anteriormente expuesto seguiremos utilizando en este curso los códigos propios de error de MySQL.

Para saber el código de un error, se puede averiguar de varias formas:
1. Como en otros lenguajes, provocarlo para obtener su número de error, que luego utilizaremos en el manejador del error. Ej.: Hemos visto en páginas anteriores de este curso el error que se produce cuando se intenta leer una fila de un cursor y ya existen más:
El 1329 indica el número de error de MySQL y entre paréntesis, en este caso 02000, el número SQLSTATE correspondiente.
2. Mirando el manual, suele venir una tabla detallada de errores con su número y descripción en un apéndice dedicado.

3) Condiciones de error con nombre:

3.1) **Predefinidas de MySQL:**

SQLEXCEPTION.
SQLWARNING.
NOT FOUND.

Ejemplos:

/* Si ocurre cualquier condición de error (salvo la excepción NOT FOUND tratada en el apartado de cursores) la variable v_error valdrá 1 y continuará la ejecución del programa */
DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
SET v_error=1;

/* Si ocurre cualquier condición de error (excepto la condición NOT FOUND) el procedimiento finaliza ejecutando antes la instrucción ROLLBACK (deshacer operaciones que se hubieran realizado) y otra de advertencia de la situación de error */
DECLARE EXIT HANDLER FOR SQLEXCEPTION
BEGIN
ROLLBACK;
SELECT 'Ocurrió un error. Procedimiento terminado';
END;

/* Si la instrucción FETCH en un cursor no recupera ninguna fila*/
/* De 3 formas distintas: Condición de error con nombre, SQLSTATE y código de error de MySQL /*
DECLARE CONTINUE HANDLER FOR NOT FOUND
SET v_ultima_fila=1;
DECLARE CONTINUE HANDLER FOR SQLSTATE '02000'
SET v_ultima_fila=1;
DECLARE CONTINUE HANDLER FOR 1329
SET v_ultima_fila=1;

3.2) Definidos por el usuario:

Facilitan lectura del código y por tanto el fácil mantenimiento de la aplicación.
Consiste en “bautizar” o darle un nombre a un código de error MySQL o SQLSTATE.
Si intentas ejecutar el siguiente código que forma parte de un procedimiento:

```
5 BEGIN
6   CREATE TABLE alumnos (campo1 INTEGER );
7 END $$
```

Verás que produce una situación de error, pues la tabla ya existe:

```
ERROR 1050 (42S01): Table 'alumnos' already exists
mysql>
```

En la imagen siguiente, al mismo tiempo que tratamos el error MySQL anterior le asociamos un nombre (línea 6) para facilitar su comprensión a la hora de realizar tareas como la de mantenimiento de la aplicación como se ha señalado. De esta manera ya se puede realizar la declaración del manejador de error de la línea 7.

```
5  BEGIN
6    DECLARE tabla_existente_error CONDITION FOR 1050;
7    DECLARE EXIT HANDLER FOR tabla_existente_error
8        SELECT ('Fin del programa. La tabla que intentas crear ya existe')
9            AS 'Aviso de error';
10   CREATE TABLE alumnos (campo1 INTEGER );
11 END $$
```

4 Orden de actuación del manejador

Ante varias posibilidades de ejecutarse un manejador ¿Cuál se ejecuta?
Siempre el manejador asociado al error MySQL en primer lugar (en este caso líneas 11-12 del bloque 3 siguiente). Si este no estuviera declarado y definido entonces se ejecutaría el manejador asociado al código SQLSTATE. En último lugar el manejador asociado al manejador SQLEXCEPTION.

```
7  /*.........*/
8  BEGIN
9      DECLARE EXIT HANDLER FOR SQLEXCEPTION
10         SELECT 'Ocurrió un error. Fin del programa';
11     DECLARE EXIT HANDLER FOR 1062
12         SELECT 'Clave Repetida. Código MySQL 1062';
13     DECLARE EXIT HANDLER FOR SQLSTATE '23000'
14         SELECT 'Ocurrió error  SQLSTATE 23000';
15     INSERT INTO alumnos VALUES(1,'MARIO A. CARRERA');
16 END $$
```

El orden de actuación de los manejadores puede facilitarnos una forma de trabajo en nuestros programas. Los errores de código MySQL más habituales podrán ser tratados en manejadores específicos y aquéllos que no sean considerados podrán ser atrapados en un manejador SQLEXCEPTION (equivaldría a la parte ELSE de una sentencia CASE que atraparía todo lo que no ha sido “filtrado”).
En el siguiente ejemplo, bloque 4, se declara un manejador para tratar la excepción de clave duplicada (líneas 9 a 10), lo que ocurre es que el error producido es el código de error MySQL 1048 (el primer campo de la tabla no puede ser nulo) y no el 1062 como consecuencia de la línea 13.

En esta situación se ejecuta el manejador SQLEXCEPTION (líneas 11-12).

```
7  /*.........*/
8  BEGIN
9      DECLARE EXIT HANDLER FOR 1062
10        SELECT 'Clave Repetida. Código MySQL 1062';
11     DECLARE EXIT HANDLER FOR SQLEXCEPTION
12         SELECT 'Ocurrió un error. Fin del programa';
13     INSERT INTO alumnos VALUES(NULL,'MARIO A. CARRERA');
14 END $$
```

Se podría hacer uso de una función del tipo “err_code( )” que indique el código de error que se ha producido o una variable que almacenara el código de error y su mensaje. Esta función está incluida en los lenguajes de otros gestores de bases de datos y en otros lenguajes (p.ej PHP) pero todavía no en MySQL (se espera para la versión 5.2, la especificación SQL:2003 sí que lo incluya). También se echa en falta por el momento (aparecerá en la versión 5.2) la posibilidad de que el usuario provoque intencionadamente situaciones de error (sentencias del tipo SIGNAL, RAISE…) y poder atenderlas dentro de manejadores de error propios.

5 Ámbito de actuación del manejador.

Como hemos visto los manejadores actúan en todos aquellos bloques donde se han declarado; su alcance llega también a los bloques anidados.
Ej.:

```
7  /*.........*/
8  BEGIN
9      DECLARE EXIT HANDLER FOR 1062
10        SELECT 'Clave Repetida. Código MySQL 1062';
11     DECLARE EXIT HANDLER FOR SQLEXCEPTION
12           SELECT 'Ocurrió un error. Fin del programa'
13               AS 'Aviso de error';
14     BEGIN
15        INSERT INTO alumnos VALUES(NULL,'MARIO A. CARRERA');
16     END;
17 END $$
```

Aunque la excepción se produce en el bloque interno (líneas 14 a 16), será atrapada por el manejador declarado en un nivel superior (bloque externo):

Aviso de error
Ocurrió un error. Fin del programa

Si se atrapa en un bloque interno, ya no se propaga a la del bloque superior externo:

```
7  /*.........*/
8  BEGIN
9      DECLARE EXIT HANDLER FOR 1062
10        SELECT 'Clave Repetida. Código MySQL 1062';
11     DECLARE EXIT HANDLER FOR SQLEXCEPTION
12           SELECT 'Ocurrió un error. Fin del programa'
13               AS 'Aviso de error';
14     BEGIN
15        DECLARE EXIT HANDLER FOR 1062
16           SELECT 'Bloque interno. Clave Repetida. Código MySQL 1062';
17        DECLARE EXIT HANDLER FOR SQLEXCEPTION
18           SELECT 'Bloque interno. Ocurrió un error. Fin del programa'
19               AS 'Aviso de error';
20        INSERT INTO alumnos VALUES(NULL,'MARIO A. CARRERA');
21     END;
22 END $$
```

Aviso de error
Bloque interno. Ocurrió un error. Fin del programa

6 Ejemplo de tratamiento de errores.

El siguiente ejemplo resume todo lo tratado en este apartado 2.7. No se incluye el manejador de errores NOT FOUND pues en el procedimiento no se utilizan cursores.

```
DELIMITER $$
DROP PROCEDURE IF EXISTS error8 $$
CREATE PROCEDURE error8 (p_id INT, p_alumno VARCHAR(30),
                         OUT p_error_num INT, OUT p_error_text VARCHAR(100))
MODIFIES SQL DATA
BEGIN
    DECLARE clave_repetida_error CONDITION FOR 1062;
    DECLARE clave_nula_error CONDITION FOR 1048;
    DECLARE CONTINUE HANDLER FOR clave_repetida_error
        BEGIN
          SET p_error_num=1062;
          SET p_error_text='Clave duplicada';
        END;
    DECLARE CONTINUE HANDLER FOR clave_nula_error
        BEGIN
          SET p_error_num=1048;
          SET p_error_text='Clave nula';
         END;
    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
        BEGIN
          SET p_error_num=-1;
          SET p_error_text='Ocurrió un error';
         END;
    SET p_error_num=0;
    INSERT INTO alumnos VALUES(p_id,p_alumno);
    IF p_error_num=0 THEN
         SET p_error_text='Alta de alumno realizada';
    END IF;
END $$
```