MANEJO DE ERRORES EN MYSQL

Códigos de error.

Podemos tener tres formas de definir un error:

- Con un código de error propio de MySQL.

  DECLARE CONTINUE HANDLER FOR 1062 SET duplicate_key=1;

- Con un código SQLSTATE definido por ANSI standard. Son independientes de la base de datos, lo que significa que deben tener el mismo valor de error para cualquier base de datos ANSI compatible. Así, Oracle, SQL Server, DB2, y MySQL devolverán el mismo valor de SQLSTATE (23000) cuando haya un error de primary key duplicada.

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

- Con un nombre de condición: **SQLEXCEPTION, SQLWARNING** y **NOT FOUND**.

En teoría es mejor utilizar códigos SQLSTATE porque son independientes de la plataforma y hacen que el código sea más portable. Sin embargo, hay algunas razones para utilizar mejor el código de error de MySQL, en los procedimientos almacenados:

- El lenguaje para la escritura de procedimientos almacenados en Oracle y SQL Server es totalmente incompatible con el de MySQL.

- No todos los códigos de error MySQL tienen su equivalente en código SQLSTATE. Cada error de Mysql está relacionado con un código de error SQLState, pero no siempre esta relación es uno a uno. 'HY000' es un código de error SQLSTATE para propositos generales que devuelve MySQL cuando su código de error no tiene asociado un código SQLSTATE.

En los manuales de la página oficial de MySql están sus códigos de error. A continuación se indican algunos de ellos.

Código MySQL	Código SQLSTATE	Mensaje de error
1011	HY000	Error on delete of ‘%s’ (errno: %d)
1021	HY000	Disk full
1027	HY000	Locked against change
1036	HY000	Table is read only
1062	23000	Duplicate entry for key
1099	HY000	Table was locked with a READ lock and can’t be updated
1106	42000	Incorrect parameters to procedure
1146	42S02	Table doesn't exist
1114	HY000	The table is full
1242	21000	Subquery returns more than 1 row
1264	22003	Out of range for column **numeric**
1265	1000	Data truncated for column
1317	70100	Query execution was interrupted
1325	24000	Cursor is already open
1329	2000	No data to FETCH
1338	42000	Cursor declaration after handler declaration
1348 1357	HY000	Column is not updatable
1406	22001	Data too long for column varchar

Detección de errores y tratamiento. HANDLER.

Durante la ejecución de un programa pueden ocurrir errores que requieran un manejo especial como puede ser salir de la secuencia de programa actual o continuar la ejecución. Para ello, MySql tiene los manejadores o HANDLER.

```
DECLARE acción HANDLER
    FOR valor_condición [, valor_condición] ...
    instrucción/es;

acción:
    CONTINUE
  | EXIT
  | UNDO

valor_condición:
    código_error_Mysql
  | SQLSTATE [VALUE] sqlstate_value
  | nombre_condición
  | SQLWARNING
  | NOT FOUND
  | SQLEXCEPTION
```

La instrucción **DECLARE ... HANDLER** especifica un controlador para una o más condiciones de error. Si una de esas condiciones ocurre, las instrucciones especificadas se ejecutan. Pueden ser una sola instrucción como **SET nombre_variable = *valor***, o un conjunto de ellas escritas entre **BEGIN y END**.

Las declaraciones de los Handler deben aparecer después de las declaraciones de las variables y de los cursores.

Acción es un valor que indica qué hará el handler una vez ocurrido el error y ejecutadas la instrucción/es del mismo:

- **CONTINUE**
- **EXIT**
- **UNDO**: No soportado

Valor_condición indica la condición o tipo de condiciones que activan el handler y pueden ser:

- Un código de error de MySQL.

- Un valor SQLSTATE. Dependiendo de cuales sean los 2 primeros caracteres del código de error SQLSTATE:

`'00.....'`	Ausencia de error.
`'01.....'`	Error Warning.
`'02.....'`	Error Not Found.
`>'02.....'`	Excepciones.

- **`nombre_condición`** especificado previamente con **`DECLARE ... CONDITION`**. Se explica posteriormente.
- **`SQLWARNING`** es la abreviatura para todos los códigos de SQLSTATE que comienzan por `'01'`
- **`NOT FOUND`** es la abreviatura de la clase de los valores de SQLSTATE que comienzan con `'02'`. Se usa para comprobar el final de los cursores (SQLSTATE = '02000') o también para los SELECT ... INTO que no recuperan ninguna fila.
- **`SQLEXCEPTION`** es la abreviatura de la clase de los valores de SQLSTATE que son excepciones.

Controlador para comprobar la existencia de una tabla. Si no existe, el código de error de MySQL es 1146:

```
BEGIN
  declare a int default 0;
  declare continue handler for 1146      -- ó set a=1;
      BEGIN
         set a=1;
      END;
  select * from tabla;    -- esta tabla no existe
  select a;               -- continúa la ejecución y muestra a=1

END;
```

Controlador para comprobar si una SELECT INTO devuelve alguna fila:

```
BEGIN
  declare a int;
  declare exit handler for NOT FOUND
      BEGIN
       select "la select no devuelve ninguna fila";
      END;
  select idtabla into a from tabla where idtabla=1;
  select a;            -- No se visualizaría el valor de a
END;
```

El siguiente ejemplo usa un handler para el error SQLSTATE '23000' que ocurre cuando se duplica una clave.

```
delimiter //
CREATE PROCEDURE handlerdemo ()
BEGIN
  DECLARE CONTINUE HANDLER FOR SQLSTATE '23000' SET @x2 = 1;
```

```
  SET @x = 1;
  INSERT INTO test.t VALUES (1);
  SET @x = 2;
  INSERT INTO test.t VALUES (1);
  SET @x = 3;
END //
DELIMITER ;
```

```
CALL handlerdemo();

SELECT @x,@x2;    -- Muestra los valores 3 y 1
```

Para ignorar una condición de error, se declara un HANDLER con CONTINUE asociado a un bloque vacío.

```
DECLARE CONTINUE HANDLER FOR SQLWARNING begin end;
```

Si tuviésemos 3 Handler para controlar un determinado error, cuando éste se produjera se ejecutaría el handler con error de MySql, si éste no existiera el handler con error de SQLSTATE y de no existir ninguno de los dos anteriores el error sería capturado por el handler con SQLEXCEPTION.

```
declare exit handler for sqlstate='23000' select “error de SqlState”;
declare exit handler for sqlexception select “error inesperado”;
declare exit handler for 1062 select “error de Mysql”;
```

En este ejemplo el que capturaría el error sería el handler del error de MySql.

Por otro lado es importante saber que:

- Las órdenes INSERT, UPDATE y DELETE no dan ningún error cuando no afectan a ninguna fila.

- La orden SELECT .... INTO genera el error de MySql 1172 (Return consisted of more than one row) cuando devuelve más de una fila.

- La orden SELECT .... INTO si no devuelve ninguna fila en MySql no muestra ningún error y continúa en secuencia por lo que podemos optar por:

  - Declarar para capturar el error un handler para NOT FOUND.

  - Preguntar si la variable o variables del INTO son nulas.

```
BEGIN
   declare a int;
   select idtabla into a from tabla where idtabla=1;
   if a is NULL then
       select “la select no devuelve ninguna fila”;
   else
        select a;
   end if;
```

```
END;
```

Declare...Condition

```
DECLARE nombre_condición CONDITION FOR valor_condición

valor_condición:
    código_error_Mysql
  | SQLSTATE [VALUE] sqlstate_value
```

Declarando una condición para el ejemplo anterior que comprobaba la existencia de una tabla.

```
BEGIN
   declare a int default 0;
   declare no_existe_tabla CONDITION FOR 1146;
   declare CONTINUE HANDLER FOR no_existe_tabla
       BEGIN
        set a=1;
       END;
   select * from tabla;
   select a;
END;
```

De la misma forma pero haciéndolo con el valor de SQLSTATE en vez de con el código de error MySQL:

```
declare no_existe_tabla CONDITION FOR SQLSTATE '42S02';
```

SIGNAL

Es la manera que se tiene para levantar un error en el momento que se necesite.

```
SIGNAL valor_condición
    [SET MESSAGE_TEXT= cadena o variable
    [,MYSQL_ERRNO = valor].......

valor_condición:
    SQLSTATE valor de Sqlstate
  | nombre_condición
```

El valor_condición indica el valor del error que se devuelve, puede ser un valor SQLSTATE o el nombre de una condición definida con `DECLARE ... CONDITION`.

Los valores que toma SQLSTATE dependiendo de si queremos levantar un error WARNING, NOT FOUND o una excepción son:

SQLSTATE	ERROR MYSQL
'01000'	**Error: 1642** SQLSTATE: **01000** (ER_SIGNAL_WARN) Message: Unhandled user-defined warning condition
'02000'	**Error: 1643** SQLSTATE: **02000** (ER_SIGNAL_NOT_FOUND) Message: Unhandled user-defined not found condition
'45000'	**Error: 1644** SQLSTATE: **HY000** (ER_SIGNAL_EXCEPTION) Message: Unhandled user-defined exception condition

El siguiente procedimiento señala un error o un warning dependiendo del valor de un parámetro de entrada:

```
DELIMITER $$
CREATE PROCEDURE p (pval INT)
BEGIN
  DECLARE a int default 0;
  DECLARE especial CONDITION FOR SQLSTATE '02000';   -- ERROR NOT FOUND
  IF pval = 0 THEN
          SIGNAL SQLSTATE '01000'        -- ERROR WARNING
          SET MESSAGE_TEXT = 'Error Warning de mysql 1642';
      ELSEIF pval = 1 THEN
          SIGNAL SQLSTATE '45000'        -- EXCEPCIÓN
          SET MESSAGE_TEXT = 'Ha ocurrido el error de mysql 1644';
      ELSE
          SIGNAL especial
          SET MESSAGE_TEXT = 'Ha ocurrido el error de mysql 1643',
          MYSQL_ERRNO = 1001;

  END IF;
  select a;
END $$
DELIMITER ;
```

Si pval es 0, p() señala un warning porque el valor de SQLSTATE values comienza con '01'. Los avisos o warnings no hacen terminar el procedimiento, entonces **se visualizará el valor de a** puesto que hemos forzado un aviso y la ejecución continúa. En la consola no aparece el mensaje `'Error Warning de mysql 1642'`, para visualizarlo se ejecuta el comando **SHOW WARNINGS;**. De la misma forma con **SHOW ERRORS;** se muestran los errores que no hayan sido avisos de la última ejecución aunque también aparecen en la pantalla de Output.

Si pval es 1, p() señala una excepción ya que comienza por >02 y coloca en la variable MESSAGE_TEXT la información del mismo. El contenido de la variable a no se muestra porque al no ser un error WARNING, se acaba la ejecución.

Si pval es 2, el error es un NOT FOUND ya que comienza con 02. El valor de SQLSTATE está

especificado en este caso mediante el nombre de una condición (especial) declarada al comienzo. Se comporta igual que el caso anterior acabándose la ejecución. Como se ha modificado el número de error de Mysql, en vez de:

Error Code: 1643. Ha ocurrido un error de mysql 1643

Se muestra:

Error Code: 1001. Ha ocurrido un error de mysql 1643

```
CREATE PROCEDURE P (divisor INT)
BEGIN
  IF divisor = 0 THEN
    SIGNAL SQLSTATE '22012'
    SET MESSAGE_TEXT = 'división por 0';
  END IF;
 END;
```

Muestra **Error Code: 1644.** división por 0

```
CREATE PROCEDURE p (divisor INT)
  BEGIN
  DECLARE divide_by_zero CONDITION FOR SQLSTATE '22012';
  IF divisor = 0 THEN
    SIGNAL divide_by_zero;
  END IF;
 END;
```

Muestra **Error Code: 1644.** Unhandled user-defined exception condition

En el código siguiente se declaran 2 condiciones de error, una en el bloque principal y la otra en un bloque interno a éste. Se levanta el error del bloque interno mostrando el mensaje correspondiente.

Error Code: 1644. división por 0

El error del bloque externo no se levantará nunca porque el error del bloque interno es una excepción y se acaba el procedimiento. Si el error del bloque interno fuera un Warning, éste no se mostraría y sería sustituido por el error del bloque principal obteniéndose esta salida.

Error Code: 1644. Unhandled user-defined exception condition

Este código también mostraría el mensaje anterior cuando divisor sea distinto de 0 porque my_error es la condición de error del bloque exterior.

```
CREATE PROCEDURE p (divisor INT)
BEGIN
  DECLARE my_error CONDITION FOR SQLSTATE '45000';
  IF divisor = 0 THEN
    BEGIN
      DECLARE my_error CONDITION FOR SQLSTATE '22012';
      SIGNAL my_error SET MESSAGE_TEXT = 'división por 0';
    END;
```

```
  END IF;
  SIGNAL my_error;
END;
```

IMPORTANTE: Si tenemos una orden **SIGNAL o RESIGNAL** dentro de un HANDLER, aunque este esté definido como CONTINUE, si el error que se captura es una EXCEPCIÓN o un NOT FOUND, el procedimiento termina y no continua en la siguiente orden a la que generó el error.

.

```
CREATE PROCEDURE p()
BEGIN
  DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
      BEGIN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'No existe la tabla taux';
      END;
  DROP TABLE taux;
  select 'Esto no se mostraría nunca';
END;
```

Error Code: 1644. No existe la tabla taux

Si la tabla no existe, el error correspondiente se activa, pero el handler destruye el error original y marca el nuevo error con el SQLSTATE '45000' y se visualiza el mensaje del handler.

En este otro ejemplo se levanta un error de que la tabla no existe y se captura en el handler siguiendo en secuencia:

```
CREATE PROCEDURE p()
BEGIN
  DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
      BEGIN
            SELECT 'La tabla taux no existe;
      END;
  END;
  DROP TABLE taux;
  select 'Esto se mostraría porque se capturó el error con el handler';
END;
```

```
DELIMITER $$
CREATE PROCEDURE AddOrderItem(
          orderNo int,
          productCode varchar(45),
          qty int,
          price double,
          lineNo int )
BEGIN
       DECLARE C INT;
       SELECT COUNT(orderNumber) INTO C FROM orders
       WHERE orderNumber = orderNo;
       IF(C = 0) THEN
```

```
                SIGNAL SQLSTATE '45000'
                SET MESSAGE_TEXT = 'Order No not found in orders table';
        END IF;
        - ...
        -- ...
END$$
DELIMITER;

CREATE PROCEDURE proceso(codemp int, fecha date)
BEGIN
   IF datedif(curdate(),fecha)<(16*365) THEN
      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT='El empleado debe tener más de 16 años';
   ELSE
      UPDATE empleado SET  fnac=fecha WHERE numemp=codemp;
      -- No devuelve error si el empleado no existe, habría que comprobarlo
   END IF;
END;
```

RESIGNAL

```
RESIGNAL valor_condición
    [SET MESSAGE_TEXT= cadena o variable
    [,MYSQL_ERRNO = valor].......

valor_condición:
    SQLSTATE valor de Sqlstate
  | nombre_condición
```

RESIGNAL es igual a SIGNAL en términos de funcionalidad y de sintaxis, excepto en que:

1. RESIGNAL se usa únicamente en la declaración de un HANDLER, en otro caso genera un mensaje de error en compilación. SIGNAL se puede usar en cualquier lugar dentro de un procedimiento almacenado. El error que se muestra se si sitúa fuera de su lugar es:

   Error Code: 1645: Resignal when handler not active

2. Pueden omitirse todos los atributos RESIGNAL incluso el valor de SQLSTATE. En este caso RESIGNAL; significa " transmite el error sin cambio" lo que en Java sería equivalente al throw.

Como se indicó en el SIGNAL, un HANDLER con RESIGNAL aunque esté definido como CONTINUE no continuará la ejecución en la siguiente línea a la que produjo el error a no ser que el error capturado sea un WARNING.

```
CREATE PROCEDURE proc0()
BEGIN
   DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
      BEGIN
        select 'He capturado el error de proc1';
      END;
   CALL proc();
   SELECT 'Fin proc0';    - esta salida sí se mostraría
END;
```

```
CREATE PROCEDURE proc()
BEGIN
   DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
      BEGIN
        RESIGNAL;
      END;
   SELECT fun();
 SELECT 'Fin de proc'; -- esta salida no se  mostraría al hacer resignal
 END;                  -- en el handler termina la ejecución porque el error
                       -- capturado no es un Warnig es una exception
```

```
CREATE FUNCTION fun() RETURNS BOOLEAN
BEGIN
 SIGNAL SQLSTATE '45001'SET message_text='error función';
 RETURN TRUE;
END;
```

Mostraría

El procedimiento que llama a la función debe tener un Handler que capture el error generado por la misma. En este Handler, la orden RESIGNAL sin argumentos, mostraría Error Code: 1644. error función. La orden Select 'Fin' no se ejecutaría en este caso aunque el Handler es Continue porque el error mostrado es una excepción.

```
CREATE PROCEDURE Divide(numerador INT, denominador INT, OUT resultado double)
BEGIN
       DECLARE division_by_zero CONDITION FOR SQLSTATE '22012';

       DECLARE CONTINUE HANDLER FOR division_by_zero
             RESIGNAL SET MESSAGE_TEXT = 'Division entre cero';

       IF denominador = 0 THEN
             SIGNAL division_by_zero;
       ELSE
             SET resultado = numerador / denominador;
       END IF;
END;
```

```
CREATE PROCEDURE test_error()
BEGIN
      DECLARE CONTINUE HANDLER FOR 1146    - - la tabla t no existe
            BEGIN
                  IF !@oculto_error THEN
                        RESIGNAL;
                  END IF;
            END;
```

```
      SET @oculto_error = TRUE;
      SELECT 'el error siguiente será ignorado';
      SELECT * FROM t;
      SELECT 'el error no será ignorado';
      SET @oculto_error = FALSE;
      SELECT * FROM t;
END;



CREATE PROCEDURE prueba()
BEGIN
      DECLARE CONTINUE HANDLER FOR 1146
      BEGIN
            RESIGNAL SET MESSAGE_TEXT = 'tablita no existe';
      END;
      SELECT * FROM tablita;
END;


CALL prueba();
Error Code: 1146. tablita no existe




CREATE PROCEDURE captura_error()
BEGIN
    RESIGNAL;
END;
                              LA EJECUCIÓN DE P() DEVUELVE EL
                              ERROR 1645: Resignal when handler not active

CREATE PROCEDURE p()
BEGIN
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
      BEGIN
            CALL captura_error();
      END;
  SIGNAL SQLSTATE '45000';
END;



CREATE PROCEDURE prueba(i int)
BEGIN
      DECLARE tipo_incorrecto CONDITION FOR SQLSTATE '22012';

      DECLARE EXIT HANDLER FOR tipo_incorrecto
            BEGIN
                  RESIGNAL SET MESSAGE_TEXT = 'El tipo del dato es incorrecto';
            END;
      DECLARE EXIT HANDLER FOR SQLEXCEPTION, SQLWARNING
            BEGIN
               SELECT 'Se ha producido una error o un aviso';
               rollback;
            END;
      IF i<=0 THEN
         SIGNAL tipo_incorrecto;
      END IF;
      select * from t;
END;
```

Si i<=0 dar el error 1644 con el mensaje 'El tipo de dato es incorrecto' y acaba la ejecución

independientemente de que la acción del handler fuera CONTINUE, al incluir un RESIGNAL acaba la ejecución del programa.

Si i>0 visualiza 'Se ha producido una error o un aviso' si la tabla t no existe y acaba la ejecución. Este handler es interesante colocarlo en todos los procedimientos para que si se produce cualquier error no tratado por otros handlers anteriores se muestre un mensaje y termine el procedimiento. Notar que la orden ROLLBACK es imprescindible cuando en el cuerpo del procedimiento se han efectuado operaciones INSERT, UPDATE o DELETE) sobre la base de datos que requieran deshacerlas.

```
CREATE PROCEDURE prueba(i int)
BEGIN
      DECLARE tipo_incorrecto CONDITION FOR SQLSTATE '22012';

      DECLARE CONTINUE HANDLER FOR tipo_incorrecto
            BEGIN
                  select 'El tipo del dato es incorrecto';
            END;
      DECLARE EXIT HANDLER FOR SQLEXCEPTION, SQLWARNING
            BEGIN
               SELECT 'Se ha producido una error o un aviso';
               rollback;
            END;
      IF i<=0 THEN
         SIGNAL tipo_incorrecto;
      END IF;
      select * from t;
END;
```

En este código en vez de RESIGNAL hay una select que muestra un mensaje. Si i<=0 se muestra el mensaje 'El tipo de dato es incorrecto' y continúa la ejecución haciendo la select de la tabla t.

```
      CREATE PROCEDURE prueba_diagnostico(codprod long, des varchar(30))
      BEGIN
            DECLARE sqlstate_code varchar(64) default'00000';
            DECLARE message_text varchar(64);
            DECLARE mysqL_errno int;
            DECLARE EXIT HANDLER FOR SQLEXCEPTION
              BEGIN
                  GET DIAGNOSTICS CONDITION 1
                  sqlstate_code=RETURNED_SQLSTATE,
                  mysqL_errno=MYSQL_ERRNO,
                  message_text=MESSAGE_TEXT;
                  IF sqlstate_code<>'00000' THEN
                    CASE mysql_errno
                       WHEN 1062 THEN
                          SIGNAL SQLSTATE '45000'
                          SET MESSAGE_TEXT='El producto ya existe';
                       WHEN 1264 THEN
                          SIGNAL SQLSTATE '45000'
                          SET MESSAGE_TEXT='El código de producto es largo';
                        ELSE RESIGNAL;
                    END CASE;
                 END IF;
               END;
```

```
        INSERT INTO producto VALUES (codprod, des);
        COMMIT;
END;
```