Procedimientos Almacenados en MySQL. Errores.

Manejo de errores en procedimientos almacenados (Handling errors).

Cuando se produce un error dentro de un procedimiento almacenado, es importante manejarlo adecuadamente, como continuar o salir de la ejecución del bloque de código actual y emitir un mensaje de error significativo.
MySQL proporciona una manera fácil de definir manejadores que manejan desde condiciones generales como advertencias o excepciones a condiciones específicas, por ejemplo, códigos de error específicos. Declarando un manejador
Para declarar un controlador, use la DECLARE HANDLER siguiente manera:

```
DECLARE action HANDLER FOR condition_value statement;
```

Si una condición cuyo valor coincide con el condition_value, MySQL ejecutará la statement y continuará o saldrá del bloque de código actual en función de la *action*.
La action acepta uno de los siguientes valores:

- CONTINUE: continúa la ejecución del bloque de código adjunto (BEGIN... END).
- EXIT: finaliza la ejecución del bloque de código adjunto, donde se declara el controlador.

condition_value especifica una condición particular o una clase de condiciones que activan el controlador. condition_value acepta uno de los siguientes valores:

- Un código de error MySQL.
- Un valor estándar de SQLSTATE. O puede ser una SQLWARNING NOTFOUND, NOTFOUND o SQLEXCEPTION, que es la abreviatura de la clase de valores SQLSTATE. La condición NOTFOUND se usa para un cursor o una SELECT INTO *variable_list*.
- Una condición con nombre asociada con un código de error MySQL o un valor SQLSTATE.

La *statement* (sentencia) podría ser una declaración simple o una declaración compuesta encerrada por las palabras clave BEGIN y END. Ejemplos de manejo de errores de MySQL
Tomemos algunos ejemplos de declarar manejadores.
El siguiente controlador establece el valor de la variable hasError en 1 y continúa la ejecución si se produce una SQLEXCEPTION

```
DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
SET hasError = 1;
```

El siguiente manejador revierte las operaciones anteriores, emite un mensaje de error y sale del bloque de código actual en caso de que ocurra un error. Si lo declara dentro del bloque BEGIN END de un procedimiento almacenado, finalizará el procedimiento almacenado de inmediato.

```
DECLARE EXIT HANDLER FOR SQLEXCEPTION
BEGIN
    ROLLBACK ;
    SELECT 'An error has occurred, operation rollbacked and the stored procedure was termir
END ;
```

El siguiente controlador establece el valor de la variable RowNotFound en 1 y continúa la ejecución si no hay más filas para recuperar en caso de un cursor o una SELECT INTO:

```
DECLARE   CONTINUE   HANDLER   FOR   NOT  FOUND
SET  RowNotFound = 1;
```

Si se produce un error de clave duplicada, el siguiente controlador emite un mensaje de error y continúa la ejecución.

```
DECLARE   CONTINUE   HANDLER   FOR  1062
SELECT   'Error, duplicate key occurred' ;
```

Ejemplo de controlador MySQL en procedimientos almacenados

- Primero, cree una nueva tabla llamada SupplierProducts para la demostración:

```
 CREATE   TABLE  SupplierProducts (
    supplierId  INT ,
    productId  INT ,
     PRIMARY KEY  (supplierId , productId)
);
```

La tabla SupplierProducts almacena las relaciones entre los proveedores y productos de la tabla. Cada proveedor puede proporcionar muchos productos y cada producto puede ser proporcionado por muchos proveedores. En aras de la simplicidad, no creamos tablas de Products y Suppliers , así como las claves externas en la tabla de productos del SupplierProducts .

- En segundo lugar, cree un procedimiento almacenado que inserte la identificación del producto y la identificación del proveedor en la tabla SupplierProducts :

```
 CREATE   PROCEDURE  InsertSupplierProduct(
     IN  inSupplierId  INT ,
     IN  inProductId  INT
)
 BEGIN
     -- exit if the duplicate key occurs
     DECLARE   EXIT   HANDLER   FOR  1062
     BEGIN
      SELECT   CONCAT ( 'Duplicate key (' ,inSupplierId, ',' ,inProductId, ') occurred' )
     END ;
```

```
    -- insert a new row into the SupplierProducts
    INSERT   INTO  SupplierProducts(supplierId,productId)
    VALUES (inSupplierId,inProductId);

    -- return the products supplied by the supplier id
    SELECT COUNT (*)
    FROM  SupplierProducts
    WHERE  supplierId = inSupplierId;

END $$

DELIMITER  ;
```

Cómo funciona.
El siguiente controlador de salida finaliza el procedimiento almacenado cada vez que se produce una clave duplicada (con el código 1062). Además, devuelve un mensaje de error.

```
DECLARE   EXIT   HANDLER   FOR  1062
BEGIN
    SELECT   CONCAT ( 'Duplicate key (' ,supplierId, ',' ,productId, ') occurred' )  AS  me
END ;
```

Esta declaración inserta una fila en la tabla de productos del SupplierProducts . Si se produce una clave duplicada, se ejecutará el código en la sección del controlador. 1 2

```
INSERT   INTO  SupplierProducts(supplierId,productId)
VALUES (supplierId,productId);
```

- Tercero, llame al InsertSupplierProduct() para insertar algunas filas en la tabla SupplierProducts :

```
CALL  InsertSupplierProduct(1,1);
CALL  InsertSupplierProduct(1,2);
CALL  InsertSupplierProduct(1,3);
```

- Cuarto, intente insertar una fila cuyos valores ya existan en la tabla SupplierProducts :

```
CALL  InsertSupplierProduct(1,3);
```

Aquí está el mensaje de error:

```
+-----------------------------+
| message                     |
+-----------------------------+
| Duplicate key (1,3) occurred |
+-----------------------------+
1 row in set (0.01 sec)
```

Debido a que el controlador es un controlador EXIT, la última instrucción no se ejecuta:

```
SELECT COUNT (*)
FROM  SupplierProducts
WHERE  supplierId = inSupplierId;
```

Si cambia la EXIT en la declaración del controlador a CONTINUE , también obtendrá el número de productos proporcionados por el proveedor:

```
DROP PROCEDURE IF EXISTS InsertSupplierProduct;

DELIMITER $$

CREATE PROCEDURE InsertSupplierProduct(
    IN inSupplierId INT ,
    IN inProductId INT
)
BEGIN
    -- exit if the duplicate key occurs
    DECLARE CONTINUE HANDLER FOR 1062
    BEGIN
    SELECT CONCAT ( 'Duplicate key (' ,inSupplierId, ',' ,inProductId, ') occurred' ) A
    END ;

    -- insert a new row into the SupplierProducts
    INSERT INTO SupplierProducts(supplierId,productId)
    VALUES (inSupplierId,inProductId);

    -- return the products supplied by the supplier id
    SELECT COUNT (*)
    FROM SupplierProducts
    WHERE supplierId = inSupplierId;

END $$

DELIMITER ;
```

Finalmente, llame al procedimiento almacenado nuevamente para ver el efecto del controlador CONTINUE:

```
CALL InsertSupplierProduct(1,3);
```

Aquí está la salida:

```
+----------+
| COUNT (*) |
+----------+
|        3 |
+----------+
1 row in set (0.01 sec)

Query OK, 0 rows affected (0.02 sec)
```

Precedencia de controlador MySQL

En caso de que tenga varios manejadores que manejen el mismo error, MySQL llamará al manejador más específico para manejar el error primero según las siguientes reglas:

- Un error siempre se asigna a un código de error de MySQL porque en MySQL es el más específico.
- Un SQLSTATE puede correlacionarse con muchos códigos de error de MySQL, por lo tanto, es menos específico.
- Un SQLEXCPETION o un SQLWARNING es la abreviatura de una clase de valores SQLSTATES , por lo que es el más genérico.

Según las reglas de precedencia del controlador, el controlador de código de error MySQL, el controlador SQLSTATE y SQLEXCEPTION tienen la primera, segunda y tercera prioridad.
Supongamos que tenemos tres controladores en los controladores en el procedimiento almacenado *insert_article_tags_3*:

```
 DROP   PROCEDURE   IF EXISTS  InsertSupplierProduct;

 DELIMITER  $$

 CREATE   PROCEDURE  InsertSupplierProduct(
     IN  inSupplierId  INT ,
     IN  inProductId  INT
)
 BEGIN
     -- exit if the duplicate key occurs
     DECLARE   EXIT   HANDLER   FOR  1062  SELECT   'Duplicate keys error encountered'  Mes
     DECLARE   EXIT   HANDLER   FOR   SQLEXCEPTION   SELECT   'SQLException encountered'  N
     DECLARE   EXIT   HANDLER   FOR   SQLSTATE   '23000'   SELECT   'SQLSTATE 23000'  Error

     -- insert a new row into the SupplierProducts
     INSERT   INTO  SupplierProducts(supplierId,productId)
     VALUES (inSupplierId,inProductId);

     -- return the products supplied by the supplier id
     SELECT COUNT (*)
     FROM  SupplierProducts
     WHERE  supplierId = inSupplierId;

 END $$

 DELIMITER  ;
```

Llame al procedimiento almacenado para insertar una clave duplicada:

```
CALL  InsertSupplierProduct(1,3);
```

Aquí está la salida: 8

```
+---------------------------------+
| Message                         |
+---------------------------------+
| Duplicate keys error encountered |
+---------------------------------+
1 row in set (0.00 sec)
```

```
Query OK, 0 rows affected (0.01 sec)
```

Como puede ver, se llama al controlador de código de error MySQL.

Usar una condición de error con nombre

Comencemos con una declaración de controlador de errores.

```
DELIMITER  $$

CREATE   PROCEDURE  TestProc()
BEGIN

    DECLARE   EXIT   HANDLER   FOR  1146
    SELECT   'Please create table abc first'  Message;

    SELECT  *  FROM  abc;
END $$

DELIMITER  ;
```

¿Qué significa realmente el número 1146? Imagine que ha almacenado procedimientos contaminados con estos números en todos los lugares; Será difícil entender y mantener el código.
Afortunadamente, MySQL proporciona la sentencia DECLARE CONDITION que declara una condición de error con nombre, que se asocia con una condición.
Aquí está la sintaxis de la declaración DECLARE CONDITION :

```
DECLARE  condition_name  CONDITION   FOR  condition_value;
```

condition_value puede ser un código de error de MySQL como 1146 o un valor SQLSTATE. El condition_value está representado por el condition_name.

- Después de la declaración, puede hacer referencia a condition_name lugar de condition_value.

Entonces puede reescribir el código anterior de la siguiente manera:

```
DROP   PROCEDURE   IF EXISTS  TestProc;

DELIMITER  $$

CREATE   PROCEDURE  TestProc()
BEGIN
    DECLARE  TableNotFound  CONDITION   for  1146;

    DECLARE   EXIT   HANDLER   FOR  TableNotFound
    SELECT   'Please create table abc first'  Message;
    SELECT  *  FROM  abc;
END $$

DELIMITER  ;
```

Como puede ver, el código es más obvio y legible que el anterior. Tenga en cuenta que la declaración de condición debe aparecer antes de las declaraciones de controlador o cursor.

Raising errors: Elevar condiciones de error con las sentencias SIGNAL/RESIGNAL.

Sentencia SIGNAL

Se utiliza la instrucción SIGNAL para devolver un error o una condición de advertencia al que llama a un programa almacenado, (procedimiento almacenado, función almacenada, trigger (disparador) o evento. La instrucción SIGNAL proporciona control sobre qué información se va devolver, como el valor y el mensaje SQLSTATE.
Lo siguiente ilustra la sintaxis de la declaración SIGNAL:

```
SIGNAL   SQLSTATE  | condition_name;
SET  condition_information_item_name_1 = value_1,
   condition_information_item_name_1 = value_2, etc;
```

Después de la palabra clave SIGNAL hay un valor SQLSTATE o un nombre de condición declarado por la declaración DECLARE CONDITION. Observe que la instrucción SIGNAL siempre debe especificar un valor SQLSTATE o una condición con nombre que se define con un valor SQLSTATE. Para proporcionar información a la persona que llama, utiliza la cláusula SET. Si desea devolver varios nombres de elementos de información de condición con valores, debe separar cada par nombre/valor con una coma.
La *condition_information_item_name* puede ser MESSAGE_TEXT, MYSQL_ERRORNO, CURSOR_NAME, etc.

Ejemplo con SIGNAL.

El siguiente procedimiento almacenado agrega una línea de pedido en un pedido de ventas existente. Emite un mensaje de error si el número de pedido no existe.

```
DELIMITER  $$

CREATE   PROCEDURE  AddOrderItem(
             in  orderNo  int ,
             in  productCode  varchar (45),
             in  qty  int ,
             in  price  double ,
             in  lineNo  int  )
BEGIN
    DECLARE C INT ;

    SELECT COUNT (orderNumber)  INTO  C
    FROM  orders
    WHERE  orderNumber = orderNo;

    -- check if orderNumber exists
    IF (C != 1)  THEN
        SIGNAL   SQLSTATE   '45000'
        SET  MESSAGE_TEXT =  'Order No not found in orders table' ;
    END IF ;
    -- more code below
    -- ...
END
```

- Primero, cuenta las órdenes con el número de orden de entrada que pasamos al procedimiento almacenado.
- En segundo lugar, si el número de pedido no es 1, genera un error con SQLSTATE 45000 junto con un mensaje de error que dice que el número de pedido no existe en la tabla de pedidos.

Tenga en cuenta que 45000 es un valor genérico de SQLSTATE que ilustra una excepción no controlada definida por el usuario.
Si llamamos al procedimiento almacenado AddOrderItem() y pasamos un número de pedido inexistente, obtendremos un mensaje de error.

```
CALL  AddOrderItem ( 10 , 'S10_1678' , 1 , 95.7 , 1 ) ;
```

Sentencia RESIGNAL

Además de la declaración SIGNAL, MySQL también proporciona la declaración RESIGNAL utilizada para generar una condición de advertencia o error.
La declaración RESIGNAL es similar a la declaración SIGNAL en términos de funcionalidad y sintaxis, excepto que:

- Debe usar la instrucción RESIGNAL dentro de un controlador de error o advertencia, de lo contrario, recibirá un mensaje de error que dice RESIGNAL cuando el controlador no está activo". Tenga en cuenta que puede usar la instrucción SIGNAL cualquier lugar dentro de un procedimiento almacenado.
- Puede omitir todos los atributos de la instrucción RESIGNAL , incluso el valor SQLSTATE .

Si usa solo la instrucción RESIGNAL, todos los atributos son los mismos que los pasados al controlador de condiciones.

Ejemplo con RESIGNAL

El siguiente procedimiento almacenado cambia el mensaje de error antes de emitirlo a la persona que llama.

```
DELIMITER  $$

CREATE   PROCEDURE  Divide( IN  numerator  INT ,  IN  denominator  INT ,  OUT  result  doub
BEGIN
    DECLARE  division_by_zero  CONDITION   FOR   SQLSTATE   '22012' ;

    DECLARE   CONTINUE   HANDLER   FOR  division_by_zero
    RESIGNAL   SET  MESSAGE_TEXT =  'Division by zero / Denominator cannot be zero' ;
    --
    IF  denominator = 0  THEN
        SIGNAL  division_by_zero;
    ELSE
        SET  result := numerator / denominator;
    END IF ;
END
```

Llamemos al procedimiento almacenado Divide().

```
CALL  Divide ( 10 , 0 , @ result ) ;
```