**MySQL Error Handling using the Signal and Resignal Statements**

**By**[**Rob Gravelle**](https://www.databasejournal.com/feedback/https:/www.databasejournal.com/features/mysql/mysql-error-handling-using-the-signal-and-resignal-statements.html)

All programming languages incorporate some sort of error handling mechanism for handling unexpected occurrences and a graceful exit from the application if need be.  MySQL introduced the SIGNAL and RESIGNAL statements in version 5.5 (as per the SQL 2003 spec) to serve that purpose.  It allows you to raise your own error conditions from your stored procedures, triggers, and events. In today’s article we’ll learn some of the key differences between SIGNAL and RESIGNAL as well as how to utilize both.

**Basic Syntax**

Before MySQL 5.5, developers had to resort to workarounds such as deliberately referring to a nonexistent table to cause a routine to throw an error. Thankfully, SIGNAL (and RESIGNAL) may now be employed to provide error information to a handler and/or to the calling process. Moreover, SIGNAL provides some control over the error's attributes such as the error number, SQLSTATE value, and message.

Here is the basic syntax for the SIGNAL statement and an explanation of each part:

SIGNAL **SQLSTATE | *condition\_value***

    [SET ***signal\_information\_item= value\_1,***

    [, ***signal\_information\_item***] **= value\_2, etc**;]

Following the SIGNAL keyword is an SQLSTATE value or a condition name declared by a DECLARE CONDITION statement. Notice that the SIGNAL statement must always specify an SQLSTATE value or a named condition that defined with an SQLSTATE value.

The SQLSTATE value for a SIGNAL statement consists of a five character alphanumeric code.  The [full list](https://dev.mysql.com/doc/refman/5.0/en/error-messages-server.html) of pre-defined codes are listed in the docs. Never start your own SQLSTATE code with '00' because such values indicate success and are not valid for signaling an error.  If your value is invalid, a Bad SQLSTATE error occurs.  For catch-all error handling, you should assign an SQLSTATE value of '45000', which signifies an “unhandled user-defined exception.”

To provide the caller with information, you use the SET clause. If you want to return multiple condition information item names with values, you need to separate each name/value pair by a comma.

The condition\_information\_item\_name may be any of the following:

* CLASS\_ORIGIN
* SUBCLASS\_ORIGIN
* MESSAGE\_TEXT
* MYSQL\_ERRNO
* CONSTRAINT\_CATALOG
* CONSTRAINT\_SCHEMA
* CONSTRAINT\_NAME
* CATALOG\_NAME
* SCHEMA\_NAME
* TABLE\_NAME
* COLUMN\_NAME
* CURSOR\_NAME

Each condition\_information\_item\_name may be specified only once in the SET clause. Otherwise, a *Duplicate condition information item error*occurs.

**Stored Procedure SIGNAL Example**

The following stored procedure adds a video rental store to the [Sakila sample database](http://mysql-tools.com/en/downloads/mysql-databases/4-sakila-db.html).  As it turns out, both of the required values act as foreign keys to other tables.  Hence, it’s a good idea to check that both of the associated input parameters contain valid values.  That’s easily accomplished by applying the count() function to each of the target columns.  In both cases, an SQLSTATE value, use '45000' is assigned to signify an “unhandled user-defined exception.”

CREATE PROCEDURE `add\_new\_store`(

    manager\_staff\_id tinyint,

    address\_id smallint)

BEGIN

  DECLARE manager\_staff\_count INT;

  DECLARE address\_count INT;

  -- check if manager exists

  SELECT COUNT(\*) INTO manager\_staff\_count

  FROM staff

  WHERE staff\_id = manager\_staff\_id;

  IF manager\_staff\_count != 1 THEN

    SIGNAL SQLSTATE '45000'

    SET MESSAGE\_TEXT = 'manager\_staff\_id not found in staff table.';

  END IF;

  -- check if address exists

  SELECT COUNT(\*) INTO address\_count

  FROM address

  WHERE address.address\_id = address\_id;

  IF address\_count != 1 THEN

    SIGNAL SQLSTATE '45000'

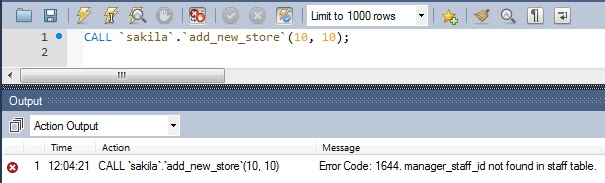
    SET MESSAGE\_TEXT = 'address\_id not found in address table.';

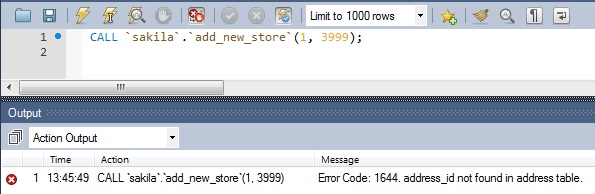
  END IF;

  -- proceed with insert...

END

Here are the corresponding error messages in the Workbench console:





**Using the RESIGNAL Statement**

The MySQL RESIGNAL statement is similar to the SIGNAL statement in terms of both functionality and syntax, except that RESIGNAL may be used to raise a warning or error condition.

RESIGNAL [**SQLSTATE | *condition\_value***]

    [SET ***signal\_information\_item*** ***= value\_1,***

    [, ***signal\_information\_item***] **= value\_2, etc**;]

Here’s our previous stored procedure rewritten to utilize the RESIGNAL statement.  This time we declare an exit handler for the SQLSTATE value of “45000”.  Inside the handler statement, we can set the MESSAGE\_TEXT dynamically based on the values of the manager\_staff and the address counts.  It might look strange that we are referencing the counts variables before they’ve been set but the exit handler does not execute until an error has occurred.

CREATE PROCEDURE `add\_new\_store\_with\_resignal`(

    manager\_staff\_id tinyint,

    address\_id smallint)

BEGIN

  DECLARE manager\_staff\_count INT;

  DECLARE address\_count INT;

  DECLARE invalid\_value CONDITION FOR SQLSTATE '45000';

  DECLARE EXIT HANDLER FOR invalid\_value

    IF manager\_staff\_count != 1 THEN

      RESIGNAL SET MESSAGE\_TEXT = 'manager\_staff\_id not found in staff table.';

    ELSEIF address\_count != 1 THEN

      RESIGNAL SET MESSAGE\_TEXT = 'address\_id not found in address table.';

    END IF;

  -- check if manager exists

  SELECT COUNT(\*) INTO manager\_staff\_count

  FROM staff

  WHERE staff\_id = manager\_staff\_id;

  IF manager\_staff\_count != 1 THEN

    SIGNAL invalid\_value;

  END IF;

  -- check if address exists

  SELECT COUNT(\*) INTO address\_count

  FROM address

  WHERE address.address\_id = address\_id;

  IF address\_count != 1 THEN

    SIGNAL invalid\_value;

  END IF;

  -- proceed with insert...

END

**Using the SIGNAL Statement in Triggers**

Including the SIGNAL statement in your stored procedures is a great way to validate user input provided that the user is in fact using the procedure to insert or update the data.  Validation within stored procedures will provide no protection whatsoever for data that is added directly into the database via an insert or update statement.

For that reason it may be prudent to include some error handling within triggers.  For instance, here is an insert trigger that employs the same validation as the add\_new\_store procedure.

CREATE TRIGGER before\_insert\_new\_store

     BEFORE INSERT ON store FOR EACH ROW

     BEGIN

          IF (SELECT COUNT(staff\_id) FROM staff

                WHERE staff\_id = NEW.manager\_staff\_id) != 1

          THEN

               SIGNAL SQLSTATE '45000'

                 SET MESSAGE\_TEXT = 'Cannot add or update row: manager\_staff\_id not found in staff table.';

           ELSEIF (SELECT COUNT(address\_id) FROM address

                  WHERE address\_id = NEW.address\_id) != 1

          THEN

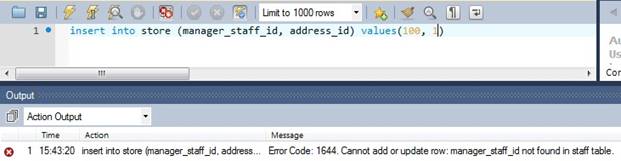
               SIGNAL SQLSTATE '45000'

                 SET MESSAGE\_TEXT = 'Cannot add or update row: address\_id not found in address table.';

          END IF;

     END;

Once again, here is the corresponding error message in the Workbench console:



**Conclusion**

The SIGNAL and RESIGNAL statements represent a giant leap forward in MySQL error handling, but they are not perfect.  The main drawback to using the SIGNAL and RESIGNAL statements right now is that they are still missing a mechanism for fetching the SQLCODE or SQLSTATE within your error handler.  Without a way to find out what the error was, you have to either code an error handler for every possible condition or be selective with which errors you catch.