MySQL - Error Handling and Exceptions

Tushar B. Kute,
http://tusharkute.com

Exception

- Exception can be said to be any abnormal condition in a program resulting to the disruption in the flow of the program.
- Whenever an exception occurs the program halts the execution and thus further code is not executed. Thus exception is that error which python script is unable to tackle with.
- Exception in a code can also be handled. In case it is not handled, then the code is not executed further and hence execution stops when exception occurs.

Signal Statement

- The SIGNAL query is a mechanism used to return a warning or error message appearing during the execution of a stored program, such as stored procedure, trigger or event, or stored function.
- This statement provides error information to an error handler, the outer portion of an application, or the client.
- It also provides control over error characteristics such as error number, SQLSTATE, value, and message in stored procedures.

Signal Statement

- The SIGNAL statement does not require any privileges for their execution.
- Syntax:
  - SIGNAL SQLSTATE | condition_name;
  - SET condition_information_item_name1 = value1, condition_information_item_name1 = value2, etc;

Signal Statement

- The SQLSTATE or a condition_name declared by the DECLARE CONDITION statement indicates the error value to be returned.
- It is to note that the SIGNAL statement must have an SQLSTATE value or a named condition defined with an SQLSTATE value.
- The SQLSTATE consists of five alphanumeric characters. We do not use the SQLSTATE code with '00' because it indicates success, which is not valid for raising an error.
- A Bad SQLSTATE error is found when the value is invalid. If we want to catch-all error handling, we must assign the SQLSTATE code '45000', which means an unhandled user-defined exception.

Signal Statement

- The condition_information_item_name can be any of the following and must be specified only once in the SET clause. Otherwise, it will return a duplicate condition information item error.
  - CLASS_ORIGIN
  - MESSAGE_TEXT
  - MYSQL_ERRNO
  - CONSTRAINT_NAME
  - SCHEMA_NAME
  - TABLE_NAME
  - CURSOR_NAME, etc.

Signal Statement Example

- Following procedure accepts the short form of the degrees and returns the full forms of them.
- If we pass a value other than B-Tech, M-Tech, BSC, MSC it generates an error message.

Signal Statement Example

```
DELIMITER //
CREATE PROCEDURE example(IN degree VARCHAR(20), OUT full_form
Varchar(50))
   BEGIN
      IF degree='B-Tech' THEN SET full_form = 'Bachelor of Technology';
      ELSEIF degree='M-Tech' THEN SET full_form = 'Master of
Technology';
      ELSEIF degree='BSC' THEN SET full_form = 'Bachelor of Science';
      ELSEIF degree='MSC' THEN SET full_form = 'Master of Science';
      ELSE
         SIGNAL SQLSTATE '01000'
      SET MESSAGE_TEXT = 'Choose from the existing values', MYSQL_ERRNO
= 12121;
         SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Given degree is not valid', MYSQL_ERRNO = 1001;
      END IF;
   END //
DELIMITER ;
```

Signal Statement Example

```
DELIMITER //
CREATE PROCEDURE example(IN degree VARCHAR(20), OUT full_form Varchar(50))
  BEGIN
    IF degree='B-Tech' THEN SET full_form = 'Bachelor of Technology';
    ELSEIF degree='M-Tech' THEN SET full_form = 'Master of Technology';
    ELSEIF degree='BSC' THEN SET full_form = 'Bachelor of Science';
    ELSEIF degree='MSC' THEN SET full_form = 'Master of Science';
    ELSE
      SIGNAL SQLSTATE '01000'
    SET MESSAGE_TEXT = 'Choose from the existing values', MYSQL_ERRNO = 12121;
      SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Given degree is not valid', MYSQL_ERRNO = 1001;
    END IF;
  END //
DELIMITER ;
```

Signal Statement Example

- You can call the above procedure as shown below –

- CALL example('BSC', @fullform);

  You can retrieve the value of the variable using SELECT statement –

- mysql> SELECT @fullform;

  If you pass an invalid value to the procedure, it will generate an error message as follow –

- mysql> CALL example ('BBC', @fullform);

Example:

```
DELIMITER //
CREATE PROCEDURE example (num INT)
  BEGIN
    DECLARE testCondition CONDITION FOR SQLSTATE '45000';
    IF num < 0 THEN
      SIGNAL SQLSTATE '01000';
    ELSEIF num > 150 THEN
      SIGNAL SQLSTATE '45000';
    END IF;
  END //
DELIMITER ;
```

Example:

- You can call the above procedure by passing two values as shown below –

```
mysql> DELIMITER ;
mysql> CALL example(15);
Query OK, 0 rows affected (0.00 sec)

mysql> CALL example(160);
ERROR 1644 (45000): Unhandled user-defined exception condition
```

The RESIGNAL statement

- MySQL provides RESIGNAL statement for raising a warning or error condition similar to the SIGNAL statement in terms of functionality and syntax, except that:
- The RESIGNAL statement must be used within an error or warning handler themselves. Otherwise, MySQL generates an error message: RESIGNAL when the handler is not active.
- The RESIGNAL statement can be used without any attributes, even the SQLSTATE value or attributes as in the SIGNAL statement.
- If we use only RESIGNAL statement in the stored program, all attributes are the same as those passed to the condition handler.

The RESIGNAL statement

```
DELIMITER $$
CREATE PROCEDURE getDivision (IN numerator INT, IN denominator INT, OUT res double)
BEGIN
   DECLARE Division_By_Zero CONDITION FOR SQLSTATE '45000';
   DECLARE CONTINUE HANDLER FOR Division_By_Zero
   RESIGNAL SET MESSAGE_TEXT = 'The denominator cannot be zero';
   --
   IF denominator = 0 THEN
      SIGNAL Division_By_Zero;
   ELSE
      SET res := numerator / denominator;
   END IF;
END $$
DELIMITER ;
```

The RESIGNAL statement

```
DELIMITER $$
CREATE PROCEDURE getDevision (IN numerator INT, IN denominator INT, OUT
res double)
BEGIN
    DECLARE Division_By_Zero CONDITION FOR SQLSTATE '45000';
    DECLARE CONTINUE HANDLER FOR Division_By_Zero
    RESIGNAL SET MESSAGE_TEXT = 'The denominator cannot be zero';
    --
    IF denominator = 0 THEN
        SIGNAL Division_By_Zero;
    ELSE
        SET res := numerator / denominator;
    END IF;
END $$
DELIMITER ;
```

Check the output

- mysql> CALL getDivision (25, 0, @res);
  ERROR 1644 (45000): The denominator cannot be zero

- mysql> CALL getDivision (25, 3, @res);
  Query OK, 0 rows affected (0.04 sec)

Handler Actions

- While working with stored procedures in MySQL if an exception or occurs the execution of the procedure terminates abruptly, to avoid this you need to handle the exceptions in MYSQL.
- MySQL provides a handler to handle the exceptions in the stored procedures.
- You can handle these exceptions by declaring a handler using the MySQL DECLARE ... HANDLER Statement.

Handler Actions

- Syntax:

```
DECLARE handler_action HANDLER
  FOR condition_value
  statement
```

Handler Actions

- The handler_action
- The handler_action is the action to be performed when the given condition(s) are satisfied.
- You can provide the following as values for handler actions.
  - CONTINUE – The current program will continue execution of the procedure.
  - EXIT – This terminates the execution of the procedure.
  - UNDO – InnoDB does not support this action.

Handler Actions

- The condition_value is the condition to be satisfied, you can pass multiple condition values. You can provide the following as values for condition value.
  - mysql_error_code – This is an integer literal indicating the error code.
  - sqlstate_value – This is a 5-character string literal specifying the SQLSTATE value.
  - condition_name – The name of the user defined condition specified with DECLARE ... CONDITION.
  - SQLWARNING – Shorthand value for SQLSTATE starts with '01'.
  - NOT FOUND – Shorthand value for SQLSTATE starts with '02'.
  - SQLEXCEPTION – Short hand value to specify the exception.

Example: Create a table

- Assume we have created a table with name tutorials in MySQL database using CREATE statement as shown below –

```
CREATE TABLE lecture (
  ID INT PRIMARY KEY,
  TITLE VARCHAR(100),
  AUTHOR VARCHAR(40),
  DATE VARCHAR(40)
);
```

Example: Insert some values

- insert into lecture values(1, 'Java', 'Krishna', '2019-09-01');
- insert into lecture values(2, 'JFreeCharts', 'Satish', '2019-05-01');
- insert into lecture values(3, 'JavaSprings', 'Amit', '2019-05-01');
- insert into lecture values(4, 'Android', 'Ram', '2019-03-01');
- insert into lecture values(5, 'Cassandra', 'Pruthvi', '2019-04-06');

Example: Create table for backup

- Let us create another table to back up the data –

```
CREATE TABLE backup (
  ID INT,
  TITLE VARCHAR(100),
  AUTHOR VARCHAR(40),
  DATE VARCHAR(40)
);
```

Create cursor function

```
DELIMITER &&
CREATE PROCEDURE ExampleProc()
  BEGIN
    DECLARE done INT DEFAULT 0;
    DECLARE lectureID INTEGER;
    DECLARE lectureTitle, lectureAuthor, lectureDate VARCHAR(20);
    DECLARE cur CURSOR FOR SELECT * FROM lecture;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
    OPEN cur;
    label: LOOP
    FETCH cur INTO lectureID, lectureTitle, lectureAuthor, lectureDate;
    INSERT INTO backup VALUES(lectureID, lectureTitle, lectureAuthor, lectureDate);
    IF done = 1 THEN LEAVE label;
    END IF;
    END LOOP;
    CLOSE cur;
  END&&
DELIMITER ;
```

Check the output

- You can call the above procedure as shown below –

```
mysql> CALL ExampleProc;
```

- If you verify the contents of the backup table you can see the inserted records as shown below –

```
mysql> select * from backup;
```

Thank you

@mitu_skillologies

@mITuSkillologies

@mitu_group

@mitu-skillologies

@MITUSkillologies

kaggle

@mituskillologies

Web Resources
https://mitu.co.in
http://tusharkute.com

@mituskillologies

contact@mitu.co.in

tushar@tusharkute.com