SIGNAL and RESIGNAL

In this lesson we will learn how to raise errors and warnings using SIGNAL and RESIGNAL statements.

We'll cover the following

SIGNAL and RESIGNAL

SIGNAL statement is a way to return an error to a handler or a client application. The sender can choose which error characteristics (like the error number or message text) to return. **SIGNAL** provides a way to handle unexpected events that may lead to application termination. By raising errors, we can provide information to the error handler for a graceful exit rather than an abrupt termination.

The **SIGNAL** keyword is used to raise errors. It is followed by a **SQLSTATE value** or a named condition defined with a SQLSTATE value. **SIGNAL** cannot be associated with a MySQL error code. The SQLSTATE value 45000 is generic and can be used to catch any un-handled user defined exceptions.

The **SET** clause is used to return information about the error. The diagnostic information about an error includes **MYSQL_ERRNO, MESSAGE_TEXT, CLASS_ORIGIN, SUBCLASS_ORIGIN, CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME, CATALOG_NAME, SCHEMA_NAME, TABLE_NAME, COLUMN_NAME**, and **CURSOR_NAME**. The sender can choose to send more than one of these items as a comma separated list.

Syntax

> SIGNAL **SQLSTATE | condition_name**
>
> SET **condition_information_item1 = val1, condition_information_item2 = val2**, ... ;

The **RESIGNAL** statement is similar to the SIGNAL statement. It is used to raise warnings and errors. **RESIGNAL** passes on the information about an error condition to a handler. The **SQLSTATE value** and the **SET** clause are optional and **RESIGNAL** can be used alone in which case it just passes the error without modifying the error information. When the **SET** clause is used with **RESIGNAL**, it is used to modify the error attributes like changing the error number or the message text. **RESIGNAL** can only be used within the scope of a condition handler while the **SIGNAL** statement can be used anywhere in a stored procedure.

Syntax

> RESIGNAL **[SQLSTATE | condition_name]**
>
> [SET **condition_information_item1 = val1, condition_information_item2 = val2**, ...];

> Connect to the terminal below by clicking in the widget. Once connected, the command line prompt will show up. Enter or copy and paste the command **./DataJek/Lessons/58lesson.sh** and wait for the MySQL prompt to start-up.

```
-- The lesson queries are reproduced below for convenient copy/paste into the terminal.

-- Query 1
DELIMITER **
CREATE PROCEDURE AddActor(
        IN Name1 VARCHAR(20),
        IN Name2 VARCHAR(20),
        IN Birthday DATE,
        IN networth INT )
BEGIN
    DECLARE age INT DEFAULT 0;
    SELECT TIMESTAMPDIFF (YEAR, Birthday, CURDATE())
```

```
    SELECT TIMESTAMPDIFF (YEAR, Birthday, CURDATE())
    INTO age;

    IF(age < 1) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Incorrect DoB value. Age cannot be zero or less than zero';
    END IF;

    IF(networth < 1) THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Incorrect NetWorth value. Net worth cannot be zero or less than ze
    END IF;
    -- If all ok then INSERT a row in the Actors table
    INSERT INTO Actors (FirstName, SecondName, Dob, NetWorthInMillions)
    VALUES(Name1, Name2, Birthday, networth);
END **
DELIMITER ;

-- Query 2
CALL AddActor('Jackson','Samuel','2020-12-21', 250);
CALL AddActor('Jackson','Samuel','1948-12-21', 0);

-- Query 3
DROP PROCEDURE AddActor;
DELIMITER **
CREATE PROCEDURE AddActor(
        IN Name1 VARCHAR(20),
        IN Name2 VARCHAR(20),
        IN Birthday DATE,
        IN networth INT)
BEGIN
    DECLARE age INT DEFAULT 0;
    DECLARE InvalidValue CONDITION FOR SQLSTATE '45000';

    DECLARE CONTINUE HANDLER FOR InvalidValue
    IF age < 1 THEN
        RESIGNAL;
    ELSEIF networth < 1 THEN
        RESIGNAL;
    END IF;

    SELECT TIMESTAMPDIFF (YEAR, Birthday, CURDATE())
    INTO age;

    IF age < 1 THEN
        SIGNAL InvalidValue;
    ELSEIF networth < 1 THEN
        SIGNAL InvalidValue;
    ELSE
        INSERT INTO Actors (FirstName, SecondName, Dob, NetWorthInMillions)
        VALUES(Name1, Name2, Birthday, networth);
    END IF;
END **
DELIMITER ;

-- Query 4
CALL AddActor('Jackson','Samuel','2020-12-21', 250);
CALL AddActor('Jackson','Samuel','1948-12-21', 0);

-- Query 5
DROP PROCEDURE AddActor;
DELIMITER **
CREATE PROCEDURE AddActor(
```

```
        IN FirstName varchar(20),
        IN SecondName varchar(20),
        IN DoB date,
        IN networth int )
BEGIN
    DECLARE age INT DEFAULT 0;
    DECLARE InvalidValue CONDITION FOR SQLSTATE '45000';

    DECLARE CONTINUE HANDLER FOR InvalidValue
        IF age < 1 THEN
            RESIGNAL SET MESSAGE_TEXT = 'Incorrect DoB value. Age cannot be zero or less than zero
        ELSEIF networth < 1 THEN
            RESIGNAL SET MESSAGE_TEXT = 'Incorrect NetWorth value. Net worth cannot be zero or les
        END IF;

    SELECT TIMESTAMPDIFF (YEAR, DoB, CURDATE())
    INTO age;

    IF age < 1 THEN
        SIGNAL InvalidValue;
    ELSEIF networth < 1 THEN
        SIGNAL InvalidValue;
    ELSE
        INSERT INTO Actors (FirstName, SecondName, Dob, NetWorthInMillions)
        VALUES(Name1, Name2, Birthday, networth);
    END IF;
END **
DELIMITER ;

-- Query 6
CALL AddActor('Jackson','Samuel','2020-12-21', 250);
CALL AddActor('Jackson','Samuel','1948-12-21', 0);
```

Terminal

1. **SIGNAL** statement is a way to validate data before it is entered in the table. The following stored procedure **AddActor** adds an actor to the **Actors** table while checking that the data is valid. **SIGNAL** statement is used to raise an error if the age or net worth values are invalid.

```
DELIMITER **

CREATE PROCEDURE AddActor(
                IN Name1 VARCHAR(20),
                IN Name2 VARCHAR(20),
                IN Birthday DATE,
                IN networth INT )
BEGIN
    DECLARE age INT DEFAULT 0;
    SELECT TIMESTAMPDIFF (YEAR, Birthday, CURDATE())
```

```
    INTO age;

    IF(age < 1) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Incorrect DoB value. Age cannot be zero o
r less than zero';
    END IF;

    IF(networth < 1) THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Incorrect NetWorth value. Net worth c
annot be zero or less than zero';
    END IF;

    -- If all ok then INSERT a row in the Actors table
    INSERT INTO Actors (FirstName, SecondName, Dob, NetWorthInMillion
s)
    VALUES(Name1, Name2, Birthday, networth);

END **
DELIMITER ;
```

The stored procedure **AddActor** takes four **IN** parameters, the actor's first and last names, date of birth and net worth. To check if the date of birth value is correct, we will find the age using the **TIMESTAMPDIFF** function. If the age comes out to be zero an error with **SQLSTATE** value 45000 is raised. 45000 represents any un-handled user defined exception and an error message explaining the error is issued to the user. Next we check the IN parameter for net worth value. The same **SQLSTATE** value 45000 is used to raise another error if the net worth value is incorrect.

Execute the following statements to call the stored procedure **AddActor** with incorrect values for **DoB** and **NetWorth** column and see how **SIGNAL** is used to raise errors:

```
CALL AddActor('Jackson','Samuel','2020-12-21', 250);

CALL AddActor('Jackson','Samuel','1948-12-21', 0);
```

2. We have used **SIGNAL** to raise an error when the age or net worth values were incorrect. Now we will write a handler to catch the error. The

were incorrect. Now we will write a handler to catch the error. The **RESIGNAL** statement is used with an error handler.

```
DROP PROCEDURE AddActor;

DELIMITER **
CREATE PROCEDURE AddActor(
              IN Name1 VARCHAR(20),
              IN Name2 VARCHAR(20),
              IN Birthday DATE,
              IN networth INT)
BEGIN
    DECLARE age INT DEFAULT 0;
    DECLARE InvalidValue CONDITION FOR SQLSTATE '45000';

    DECLARE CONTINUE HANDLER FOR InvalidValue
    IF age < 1 THEN
        RESIGNAL;
    ELSEIF networth < 1 THEN
        RESIGNAL;
    END IF;

    SELECT TIMESTAMPDIFF (YEAR, Birthday, CURDATE())
    INTO age;

    IF age < 1 THEN
        SIGNAL InvalidValue;
    ELSEIF networth < 1 THEN
        SIGNAL InvalidValue;
    ELSE
        INSERT INTO Actors (FirstName, SecondName, Dob, NetWorthInMil
lions)
        VALUES(Name1, Name2, Birthday, networth);
    END IF;
END **

DELIMITER ;
```

```
CALL AddActor('Jackson','Samuel','1948-12-21',0);

CALL AddActor('Jackson','Samuel','2019-12-21',250);
```

When **RESIGNAL** is used alone, it just passes on the error as it is. The original error message "Unhandled user-defined exception condition" is

original error message "Unhandled user-defined exception condition" is shown. Next we will show how to modify this error message.

3. Now we will modify our handler to change the **MESSAGE_TEXT** attribute of the error. We will create an error message that clearly explains the error to the user.

```
DROP PROCEDURE AddActor;

DELIMITER **
CREATE PROCEDURE AddActor(
            IN FirstName varchar(20),
            IN SecondName varchar(20),
            IN DoB date,
            IN networth int )
BEGIN
    DECLARE age INT DEFAULT 0;
    DECLARE InvalidValue CONDITION FOR SQLSTATE '45000';

    DECLARE CONTINUE HANDLER FOR InvalidValue
        IF age < 1 THEN
            RESIGNAL SET MESSAGE_TEXT = 'Incorrect DoB value. Age can
not be zero or less than zero';
        ELSEIF networth < 1 THEN
            RESIGNAL SET MESSAGE_TEXT = 'Incorrect NetWorth value. Ne
t worth cannot be zero or less than zero';
        END IF;

    SELECT TIMESTAMPDIFF (YEAR, DoB, CURDATE())
    INTO age;

    IF age < 1 THEN
        SIGNAL InvalidValue;
    ELSEIF networth < 1 THEN
        SIGNAL InvalidValue;
    ELSE
        INSERT INTO Actors (FirstName, SecondName, Dob, NetWorthInMil
lions)
        VALUES(Name1, Name2, Birthday, networth);
    END IF;

END **
DELIMITER ;
```

Here we have used a single handler for two different error conditions and once invoked it dynamically sets the error message based on the

once invoked it dynamically sets the error message based on the condition.

```
CALL AddActor('Jackson','Samuel','2020-12-21', 250);

CALL AddActor('Jackson','Samuel','1948-12-21', 0);
```