Conditional Statements

This lesson discusses the two conditional flow control statements supported by MySQL. These are IF and CASE.

Conditional Statements

Conditional Control statements execute code only if a condition holds true. These statements allow different actions to be taken based on different conditions. The condition can range from a literal or variable to a function that returns a value.

MySQL supports two conditional control statements **IF** and **CASE**. Both provide similar functionality and the choice between the two is a matter of personal preference, However, we will discuss situations in which choosing one type of statement may be better than the other.

IF Statement

The structure of MySQL **IF** statement is very similar to one used in other programming languages.There are three different forms of the **IF** statement depending on how many conditions are being tested.

A single **IF-THEN** block is used if some statements are to be executed based on a specific condition. In the case of MySQL, a condition can evaluate to **TRUE**, **FALSE**, or **NULL** (neither true nor false). So unlike other programming languages if a condition is not **TRUE** it does not automatically mean that it is

FALSE. The condition to execute the code is given between the **IF** and **THEN** words. If the condition holds true then statements written between the **IF-THEN** and **END IF** are executed otherwise the control moves to the next statement after the **IF** block. Multiple statements can be written in a block including calls to stored procedures, SET statements, loops and nested IFs.

```
IF Condition THEN

If_statements;

END IF;
```

The second variation of **IF** statement is used when we want to execute some statements when the condition for **IF** block is true and some other statements when the condition is false. In this case the **IF THEN ELSE** statement is used. The **ELSE** block is executed when the condition for **IF** block evaluates to both **FALSE** and **NULL**.

```
IF Condition THEN

If_statements;

ELSE

else_statements;

END IF;
```

If there are multiple conditions to test then the full syntax of the **IF** statement **IF-THEN-ELSEIF-ELSE** is used. In this syntax the statements corresponding to the first condition that holds true are executed. If no condition is found to be true, then the ELSE block is executed. There can be as many **ELSEIF** blocks as required.

```
IF Condition THEN

If_statements;

ELSEIF else-if_condition
```

> **else-if_statements;**
>
> …
>
> ELSE
>
> **else_statements;**
>
> END IF;

The following image explains the flow control:

CASE Statement

CASE statement is an alternate conditional control statement offered by MySQL. Any stored procedure that has **IF** statement can be replaced by CASE statement and vice versa. There are two forms of **CASE** statement.

The simple CASE statement is when we compare the output to multiple distinct values. The values after the **WHEN** keyword are sequentially compared and if there is a match then the statements in the corresponding **THEN** block are executed. If no match is found then the optional **ELSE** block is executed. In case the **ELSE** block does not exist and **CASE** cannot find a match, then an error message is issued.

> CASE **case_value**
>
> WHEN **case_value1** THEN **statements**
>
> WHEN **case_value2** THEN **statements**
>
> …
>
> [ELSE **else-statements**]
>
> END CASE;

The other type of CASE statement is the searched CASE statement which can handle complex conditions with multiple expressions. It is used to test conditions involving ranges. The condition after the **WHEN** keyword is conditions involving ranges. The condition after the **WHEN** keyword is evaluated and if it evaluates to TRUE then the statements in the corresponding

THEN block are executed. Similar to the simple case statement, if no condition evaluates to TRUE then the optional **ELSE** block is executed and an error message is issued if the **ELSE** block does not exist. To ignore errors an empty **BEGIN END** block can be written in the **ELSE** block.

> CASE
>
> WHEN **case_1 condition** THEN **statements**
>
> WHEN **case_2 condition** THEN **statements**
>
> ...
>
> [ELSE **else-statements**]
>
> END CASE;

The following image explains the flow control:

> 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/54lesson.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 GetMaritalStatus(
    IN  ActorId INT,
    OUT ActorStatus  VARCHAR(30))
BEGIN
    DECLARE Status VARCHAR (15);

    SELECT MaritalStatus INTO Status
    FROM Actors
    WHERE Id = ActorId;

    IF Status LIKE 'Married' THEN
        SET ActorStatus = 'Actor is married';
    END IF;
END**
DELIMITER ;


-- Query 2
CALL GetMaritalStatus(1, @status);
SELECT @status;
CALL GetMaritalStatus(5, @status);
SELECT @status;

-- Query 3
DROP PROCEDURE GetMaritalStatus;
DELIMITER **
CREATE PROCEDURE GetMaritalStatus(
    IN  ActorId INT,
    OUT ActorStatus  VARCHAR(30))
BEGIN
    DECLARE Status VARCHAR (15);

    SELECT MaritalStatus INTO Status
    FROM Actors
    WHERE Id = ActorId;

    IF Status LIKE 'Married' THEN
        SET ActorStatus = 'Actor is married';
    ELSE
        SET ActorStatus = 'Actor is not married';
    END IF;
END **
DELIMITER ;

-- Query 4
CALL GetMaritalStatus(1, @status);
SELECT @status;
CALL GetMaritalStatus(5, @status);
SELECT @status;

-- Query 5
DROP PROCEDURE GetMaritalStatus;
DELIMITER **
CREATE PROCEDURE GetMaritalStatus(
    IN  ActorId INT,
    OUT ActorStatus  VARCHAR(30))
BEGIN
    DECLARE Status VARCHAR (15);

    SELECT MaritalStatus INTO Status
    FROM Actors
    WHERE Id = ActorId;

    IF Status LIKE 'Married' THEN
        SET ActorStatus = 'Actor is married';
    ELSEIF Status LIKE 'Single' THEN
        SET ActorStatus = 'Actor is single';
    ELSEIF Status LIKE 'Divorced' THEN
        SET ActorStatus = 'Actor is divorced';
    ELSE
        SET ActorStatus = 'Status not found';
    END IF;
END **
DELIMITER ;
```

```
--Query 6
CALL GetMaritalStatus(1, @status);
SELECT @status;
CALL GetMaritalStatus(5, @status);
SELECT @status;
CALL GetMaritalStatus(6, @status);
SELECT @status;

-- Query 7
DROP PROCEDURE GetMaritalStatus;
DELIMITER **
CREATE PROCEDURE GetMaritalStatus(
        IN  ActorId INT,
        OUT ActorStatus VARCHAR(30))
BEGIN
    DECLARE Status VARCHAR (15);

    SELECT MaritalStatus INTO Status
    FROM Actors
    WHERE Id = ActorId;

    CASE Status
        WHEN 'Married' THEN
            SET ActorStatus = 'Actor is married';
        WHEN 'Single' THEN
                SET ActorStatus = 'Actor is single';
        WHEN 'Divorced' THEN
            SET ActorStatus = 'Actor is divorced';
        ELSE
            SET ActorStatus = 'Status not found';
    END CASE;
END**
DELIMITER ;

-- Query 8
CALL GetMaritalStatus(1, @status);
SELECT @status;
CALL GetMaritalStatus(5, @status);
SELECT @status;
CALL GetMaritalStatus(6, @status);
SELECT @status;

-- Query 9
DELIMITER **
CREATE PROCEDURE GetAgeBracket(
       IN ActorId INT,
       OUT AgeRange VARCHAR(30))
BEGIN
    DECLARE age INT DEFAULT 0;

    SELECT TIMESTAMPDIFF(YEAR, DoB, CURDATE())
        INTO age
        FROM Actors
    WHERE Id = ActorId;

    CASE
        WHEN age < 20 THEN
            SET AgeRange = 'Less than 20 years';
        WHEN age >= 20 AND age < 30 THEN
            SET AgeRange = '20+';
        WHEN age >= 30 AND age < 40 THEN
            SET AgeRange = '30+';
        WHEN age >= 40 AND age < 50 THEN
            SET AgeRange = '40+';
        WHEN age >= 50 AND age < 60 THEN
            SET AgeRange = '50+';
        WHEN age >= 60 THEN
            SET AgeRange = '60+';
        ELSE
            SET AgeRange = 'Age not found';
        END CASE;
END**
DELIMITER ;

-- Query 10
CALL GetAgeBracket(1, @status);
SELECT @status;
CALL GetAgeBracket(5, @status);
SELECT @status;
```

Terminal

1. To understand the working of the **IF** statement, we will create a stored procedure to test the marital status of actors in the **Actors** table and gradually add conditions to explain the three different forms of the **IF** statement.

```
DELIMITER **

CREATE PROCEDURE GetMaritalStatus(
    IN  ActorId INT,
    OUT ActorStatus  VARCHAR(30))
BEGIN
    DECLARE Status VARCHAR (15);

    SELECT MaritalStatus INTO Status
    FROM Actors
    WHERE Id = ActorId;

    IF Status LIKE 'Married' THEN
        SET ActorStatus = 'Actor is married';
    END IF;
END**

DELIMITER ;
```

The **GetMaritalStatus** stored procedure has two parameters; an input parameter **ActorID** and an output parameter which returns the marital status of the actor. A variable **Status** is created in which we fetch the marital status of the actor based on the input actor id. Then we test if the status is married and return the string ‘Actor is married’ if the condition in the **IF** block is true. If the condition is not true then **NULL** is returned.

To test our stored procedure we will call it with two different actor ids as follows:

```
CALL GetMaritalStatus(1, @status);
SELECT @status;

CALL GetMaritalStatus(5, @status);
SELECT @status;
```

In the first call, **NULL** is returned because the marital status is single and **IF** block is not executed.

2. We can modify the above stored procedure to return 'Actor is not married' if the condition in the **IF** branch is not true. To include an **ELSE** branch, we will drop the procedure first and then recreate it as follows:

```
DROP PROCEDURE GetMaritalStatus;

DELIMITER **

CREATE PROCEDURE GetMaritalStatus(
    IN  ActorId INT,
    OUT ActorStatus  VARCHAR(30))
BEGIN
    DECLARE Status VARCHAR (15);

    SELECT MaritalStatus INTO Status
    FROM Actors
    WHERE Id = ActorId;

    IF Status LIKE 'Married' THEN
        SET ActorStatus = 'Actor is married';
    ELSE
        SET ActorStatus = 'Actor is not married';
    END IF;

END **
```

```
DELIMITER ;
```

Here if the marital status of the actor does not match ‘Married’ then control goes to the **ELSE** block. To test our stored procedure we will call it with two different actor ids as follows:

```
CALL GetMaritalStatus(1, @status);
SELECT @status;

CALL GetMaritalStatus(5, @status);
SELECT @status;
```

3. We can further modify our stored procedure to test for more conditions based on the three values of the **MaritalStatus** column. As mentioned in the previous step, we need to drop the stored procedure first before recreating it. The **IF THEN ELSEIF ELSE** statement is used to test for multiple conditions as follows:

```
DROP PROCEDURE GetMaritalStatus;

DELIMITER **

CREATE PROCEDURE GetMaritalStatus(
    IN  ActorId INT,
    OUT ActorStatus  VARCHAR(30))
BEGIN
    DECLARE Status VARCHAR (15);

    SELECT MaritalStatus INTO Status
    FROM Actors
    WHERE Id = ActorId;

    IF Status LIKE 'Married' THEN
        SET ActorStatus = 'Actor is married';

    ELSEIF Status LIKE 'Single' THEN
        SET ActorStatus = 'Actor is single';

    ELSEIF Status LIKE 'Divorced' THEN
        SET ActorStatus = 'Actor is divorced';

    ELSE
    ELSE
        SET ActorStatus = 'Status not found';

    END IF;
END **

DELIMITER ;
```

In this example, the **IF** and **ELSEIF** blocks test the different values of the **MaritalStatus** column. The **ELSE** block is used to handle errors in case there is a **NULL** value in the **MaritalStatus** column since this block is executed if no match is found.

To test our stored procedure we will call it for three different actors as follows:

```
CALL GetMaritalStatus(1, @status);
SELECT @status;

CALL GetMaritalStatus(5, @status);
SELECT @status;

CALL GetMaritalStatus(6, @status);
SELECT @status;
```

4. Another way to add if else logic to the code is by using the **Simple CASE** statement as follows:

```
DROP PROCEDURE GetMaritalStatus;

DELIMITER **
CREATE PROCEDURE GetMaritalStatus(
    IN  ActorId INT,
    OUT ActorStatus VARCHAR(30))
BEGIN
    DECLARE Status VARCHAR (15);

    SELECT MaritalStatus INTO Status
    FROM Actors
    WHERE Id = ActorId;

    CASE Status
        WHEN 'Married' THEN
        WHEN 'Married' THEN
            SET ActorStatus = 'Actor is married';
        WHEN 'Single' THEN
            SET ActorStatus = 'Actor is single';
        WHEN 'Divorced' THEN
            SET ActorStatus = 'Actor is divorced';
        ELSE
            SET ActorStatus = 'Status not found';
    END CASE;
END**

DELIMITER ;
```

The **Simple CASE** statement is used to determine the marital status of an actor based on the value of the **Status** variable. To test our stored procedure we will call it with two different actor ids as follows:

```
CALL GetMaritalStatus(1, @status);
SELECT @status;

CALL GetMaritalStatus(5, @status);
SELECT @status;

CALL GetMaritalStatus(6, @status);
SELECT @status;
```

The results are the same as the previous step.

5. The **Searched CASE** is used to test for complex conditions. Let’s say we want to find the age bracket of an actor. We will create a stored procedure **GetAgeBracket()** and use the **DATEDIFF()** function to find the age of the actor as follows:

```
DELIMITER **

CREATE PROCEDURE GetAgeBracket(
       IN ActorId INT,
       OUT AgeRange VARCHAR(30))
BEGIN
    DECLARE age INT DEFAULT 0;

    SELECT TIMESTAMPDIFF(YEAR, DoB, CURDATE())
    INTO age
    FROM Actors
    WHERE Id = ActorId;
    WHERE id = ActorId;

    CASE
        WHEN age < 20 THEN
            SET AgeRange = 'Less than 20 years';
        WHEN age >= 20 AND age < 30 THEN
            SET AgeRange = '20+';
        WHEN age >= 30 AND age < 40 THEN
            SET AgeRange = '30+';
        WHEN age >= 40 AND age < 50 THEN
            SET AgeRange = '40+';
        WHEN age >= 50 AND age < 60 THEN
            SET AgeRange = '50+';
        WHEN age >= 60 THEN
            SET AgeRange = '60+';
        ELSE
            SET AgeRange = 'Age not found';
    END CASE;
END**
DELIMITER ;
```

The **ELSE** clause is executed in case the age comes out to be **NULL**. We can also use an empty BEGIN END block in ELSE to handle errors arising because of NULL values. To test our stored procedure we will call it with different actor ids as follows:

```
CALL GetAgeBracket(1, @status);
SELECT @status;

CALL GetAgeBracket(5, @status);
SELECT @status;
```

6. Both **IF** and CASE can be used interchangeably. **CASE** statement is more readable while **IF** is familiar to programmers and hence more easily understood. When using **CASE**, explicit error handling is needed for **NULL** values because failure to match a condition will result in an error.