Sukkur IBA University
Merit - Quality - Excellence

CSC-251: Database Management System

Semester IV(CS, SE) Section(A, B) (Fall 2020)
Course Instructor(s): Khalid Hussain

Lab 11: Control Structures

Objective(s):

1. Introduction to Control Structures
2. IF-THEN
3. CASE
4. LOOP
5. WHILE
6. REPEAT
7. LEAVE

1. Control Structures

MySQL supports the IF-THEN, CASE, ITERATE, LEAVE, LOOP, WHILE, and REPEAT constructs for flow control within stored programs. It also supports RETURN within stored functions.

Many of these constructs contain other statements, such constructs may be nested. For example, an IF statement might contain a WHILE loop, which itself contains a CASE statement.

MySQL does not support FOR loops.

2. IF-THEN

The IF statement has three forms: simple IF-THEN statement, IF-THEN-ELSE statement, and IF-THEN-ELSEIF- ELSE statement.

MySQL simple IF-THEN statement

The IF-THEN statement allows you to execute a set of SQL statements based on a specified condition. The following illustrates the syntax of the IF-THEN statement:

```
IF condition THEN
   statements;
END IF;
```

In this syntax:

- First, specify a condition to execute the code between the IF-THEN and END IF. If the condition evaluates to TRUE, the statements between IF-THEN and END IF will execute. Otherwise, the control is passed to the next statement following the END IF.
- Second, specify the code that will execute if the condition evaluates to TRUE.

We'll use the customer table from the sample database (classicmodels) for the demonstration

Example: 1

See the following GetCustomerLevel() stored procedure.

```
DELIMITER $$
CREATE PROCEDURE GetCustomerLevel(
   IN  pCustomerNumber INT,
   OUT pCustomerLevel  VARCHAR(20))
BEGIN
   DECLARE credit DECIMAL(10,2) DEFAULT 0;

   SELECT creditLimit
   INTO credit
   FROM customers
   WHERE customerNumber = pCustomerNumber;

   IF credit > 50000 THEN
      SET pCustomerLevel = 'PLATINUM';
   END IF;
END$$
DELIMITER ;
```

The stored procedure GetCustomerLevel() accepts two parameters: pCustomerNumber and pCustomerLevel.

The following statements call the GetCustomerLevel() stored procedure for customer 141 and show the value of the OUT parameter pCustomerLevel.

```
CALL GetCustomerLevel(141, @level);
SELECT @level;
```

MySQL IF-THEN-ELSE statement

In case you want to execute other statements when the condition in the IF branch does not evaluate to TRUE, you can use the IF-THEN-ELSE statement as follows:

Syntax:

```
IF condition THEN
statements; ELSE
   else-statements;
END IF;
```

In this syntax, if the condition evaluates to TRUE, the statements between IF-THEN and ELSE execute. Otherwise, the else-statements between the ELSE and END IF execute.

Let's modify the GetCustomerLevel() stored procedure. First, drop the GetCustomerLevel() stored procedure:

Example: 2

```
Drop procedure GetCustomerLevel;
DELIMITER $$
CREATE PROCEDURE GetCustomerLevel(
   IN  pCustomerNumber INT,
   OUT pCustomerLevel  VARCHAR(20))
BEGIN
   DECLARE credit DECIMAL DEFAULT 0;

   SELECT creditLimit
   INTO credit
   FROM customers
   WHERE customerNumber = pCustomerNumber;

   IF credit > 50000 THEN
      SET pCustomerLevel = 'PLATINUM';
   ELSE
      SET pCustomerLevel = 'NOT PLATINUM';
   END IF;
END$$

DELIMITER ;
```

In this new stored procedure, we include the ELSE branch. If the credit is not greater than 50,000, we set the customer level to NOT PLATINUM in the block between ELSE and END IF.

```
CALL GetCustomerLevel(447, @level);

SELECT @level;
```

The above statements call the stored procedure for customer number 447 and show the value of the OUT parameter pCustomerLevel.

MySQL IF-THEN-ELSEIF-ELSE statement

If you want to execute statements conditionally based on multiple conditions, you use the following IF-THEN-ELSEIF-ELSE statement:

```
IF condition THEN

statements;

ELSEIF elseif-condition THEN    elseif-

statements;

...

ELSE    else-

statements;

END IF;
```

In this syntax, if the condition evaluates to TRUE, the statements in the IF-THEN branch executes; otherwise, the next elseif-condition is evaluated.

If the elseif-condition evaluates to TRUE, the elseif-statement executes; otherwise, the next elseif-condition is evaluated.

The IF-THEN-ELSEIF-ELSE statement can have multiple ELSEIF branches.

If no condition in the IF and ELSE IF evaluates to TRUE, the else-statements in the ELSE branch will execute.

Example: 3

```
DELIMITER $$

CREATE  PROCEDURE  GetCustomerLevel(
IN  pCustomerNumber INT,
   OUT pCustomerLevel  VARCHAR(20))
BEGIN
   DECLARE credit DECIMAL DEFAULT 0;

   SELECT creditLimit
INTO credit
   FROM customers
WHERE customerNumber =
pCustomerNumber;

   IF credit > 50000 THEN
      SET pCustomerLevel =
'PLATINUM';
   ELSEIF credit <= 50000 AND
credit > 10000 THEN
      SET pCustomerLevel = 'GOLD';
   ELSE
      SET pCustomerLevel =
'SILVER';
   END IF;
END $$

DELIMITER ;
```

3. CASE

Besides the IF statement, MySQL provides an alternative conditional statement called the CASE statement for constructing conditional statements in stored procedures. The CASE statements make the code more readable and efficient.

The CASE statement has two forms: simpleCASE and searched CASE statements.

MySQL Simple CASE statement

The following is the basic syntax of the simple CASE statement:

Syntax:

```
CASE case_value
```

```
  WHEN when_value1 THEN statements
WHEN when_value2 THEN statements
  ...
  [ELSE else-statements]
END CASE;
```

In this syntax, the simple CASE statement sequentially compares the case_value is with the when_value1, when_value2, … until it finds one is equal. When the CASE finds a case_value equal to a when_value, it executes statements in the corresponding THEN clause.

If CASE cannot find any when_value equal to the case_value, it executes the elsestatements in the ELSE clause if the ELSE clause is available. Otherwise it issues an error.

Example 4:

```
DELIMITER $$
CREATE PROCEDURE GetCustomerShipping(
        IN  pCustomerNUmber INT,
        OUT pShipping VARCHAR(50))
BEGIN
   DECLARE customerCountry VARCHAR(100);
SELECT country
INTO customerCountry
FROM customers
WHERE customerNumber = pCustomerNUmber;
   CASE customerCountry
                WHEN  'USA' THEN SET pShipping = '2-day Shipping';
                WHEN 'Canada' THEN SET pShipping = '3-day Shipping';
                ELSE SET pShipping = '5-day Shipping';
        END CASE;
END$$
DELIMITER ;
```

The GetCustomerShipping() stored procedure accepts two parameters: pCustomerNumber as an IN parameter and pShipping as an OUT parameter.

In the stored procedure:

- First, select the country of the customer from the customers table by the input customer number.
- Second, use the simple CASE statement to determine the shipping time based on the country of the customer. If the customer locates in USA, the

shipping time is 2-day shipping. If the customer locates in Canada, the shipping time is 3-day shipping. The customers from other countries have 5day shipping.

Searched CASE statement

The simple CASE statement only allows you to compare a value with a set of distinct values.

Syntax:

```
CASE
   WHEN search_condition1 THEN statements
   WHEN search_condition1 THEN statements
   ...
   [ELSE else-statements]
END CASE;
```

To perform more complex matches such as ranges, you use the searched CASE statement. The searched CASE statement is equivalent to the IF statement, however, it's much more readable than the IF statement.

Example 5:

```
DELIMITER $$
CREATE PROCEDURE GetDeliveryStatus(
        IN pOrderNumber INT,
        OUT pDeliveryStatus VARCHAR(100) )
BEGIN
        DECLARE waitingDay INT DEFAULT 0;
   SELECT DATEDIFF(requiredDate, shippedDate)
        INTO waitingDay
        FROM orders
   WHERE orderNumber = pOrderNumber;

  CASE
        WHEN waitingDay = 0 THEN SET pDeliveryStatus = 'On Time';
        WHEN waitingDay >= 1 AND waitingDay < 5 THEN SET pDeliveryStatus = 'Late';
        WHEN waitingDay >= 5 THEN SET pDeliveryStatus = 'Very Late';
        ELSE SET pDeliveryStatus = 'No Information';
        END CASE;
END$$
DELIMITER ;
```

The stored procedure GetDeliveryStatus() accepts an order number as an IN parameter and returns the delivery status as an OUT parameter.

First, calculate the number of days between the required date and shipped date. Second, determine the delivery status based on the number of waiting days using the searched CASE statement:

MySQL CASE expression

MySQL CASE expression is a control flow structure that allows you to add if-else logic to a query. Generally speaking, you can use the CASE expression anywhere that allows a valid expression e.g., SELECT, WHERE and ORDER BY clauses.

The following illustrates the syntax of a simple CASE expression:

Syntax:

```
CASE value
   WHEN value1 THEN result1
   WHEN value2 THEN result2
   …
   [ELSE else_result]
END AS new_colunmName
```

The following shows the syntax of a searched CASE expression:

Syntax:

```
CASE
   WHEN value1 THEN result1
   WHEN value2 THEN result2
   …
   [ELSE else_result]
END AS new_colunmName
```

This example uses the CASE expression in the SELECT clause to return the delivery time based on the country name.

Example 6:

```
SELECT country,
CASE country
        WHEN 'USA' THEN '2-day Shipping'
```

```
        WHEN 'Canada' THEN '3-day Shipping'
        ELSE '5-day Shipping'
        END AS DeliveryTime FROM customers
WHERE customerNumber = 112;
```

MySQL CASE expression in the ORDER BY clause

The following example uses the CASE expression inside order by clause to sort customers by states if the state is not NULL, or sort the country in case the state is NULL:

Example 7:

```
SELECT customerName, state, country
FROM customers
ORDER BY (
   CASE
        WHEN state IS NULL THEN country
        ELSE state
        END);
```

MySQL CASE expression with an Aggregate Function

We can also use the CASE expression with an aggregate function. The following example uses the CASE expression with the SUM() function to calculate the total of sales orders by order status.

Example 8:

```
SELECT SUM(CASE
        WHEN status = 'Shipped' THEN 1
        ELSE 0
   END) AS 'Shipped',
   SUM(CASE
      WHEN status = 'On Hold' THEN 1
      ELSE 0
   END) AS 'On Hold',
   SUM(CASE
      WHEN status = 'In Process' THEN 1
      ELSE 0
   END) AS 'In Process',
   SUM(CASE
      WHEN status = 'Resolved' THEN 1
      ELSE 0
   END) AS 'Resolved',
   SUM(CASE
      WHEN status = 'Cancelled' THEN 1
      ELSE 0
   END) AS 'Cancelled',
   SUM(CASE
      WHEN status = 'Disputed' THEN 1
      ELSE 0
   END) AS 'Disputed',
   COUNT(*) AS Total FROM
   orders;
```

4. Loop

The LOOP statement allows you to execute one or more statements repeatedly.

Here is the basic syntax of the LOOP statement:

Syntax:

```
[begin_label:] LOOP     statement_list
END LOOP [end_label]
```

The LOOP can have optional labels at the beginning and end of the block.

The LOOP executes the statement_list repeatedly. The statement_list may have one or more statements, each terminated by a semicolon (;) statement delimiter.

Typically, you terminate the loop when a condition is satisfied by using the LEAVE statement.

This is the typical syntax of the LOOP statement used with LEAVE statement:

Syntax:

```
[label]: LOOP
   ...
   -- terminate the loop
   IF condition THEN
      LEAVE [label];
   END IF;
   ...
END LOOP [label];
```

The LEAVE statement immediately exits the loop. It works like the break statement in other programming languages like PHP, C/C++, and Java.

In addition to the LEAVE statement, you can use the ITERATE statement to skip the current loop iteration and start a new iteration. The ITERATE is similar to the continue statement in PHP, C/C++, and Java.

Example 9:

```
DELIMITER $$
CREATE PROCEDURE LoopDemo()
BEGIN
        DECLARE x INT ;
        DECLARE str VARCHAR(255);
        SET x = 1;
        SET str =  '';

        loop_label:  LOOP
                IF  x > 10 THEN
                        LEAVE  loop_label;
                END  IF;

                SET  x = x + 1;
                IF  (x mod 2) THEN
                        ITERATE  loop_label;
                ELSE
                        SET  str = CONCAT(str,x,',');
                END  IF;
        END LOOP loop_label;
        SELECT str;
```

```
END$$
DELIMITER ;
```

In this example:

- The stored procedure constructs a string from the even numbers e.g., 2, 4, and 6.
- The loop_label before the LOOP statement for using with the ITERATE and LEAVE statements.
- If the value of x is greater than 10, the loop is terminated because of the LEAVE statement.
- If the value of the x is an odd number, the ITERATE ignores everything below it and starts a new loop iteration.
- If the value of the x is an even number, the block in the ELSE statement will build the result string from even numbers.

5. While

The WHILE loop is a loop statement that executes a block of code repeatedly as long as a condition is true.

Syntax:

```
[begin_label:] WHILE search_condition DO
statement_list END WHILE [end_label]
```

In this syntax:

- First, specify a search condition after the WHILE keyword. The WHILE checks the search_condition at the beginning of each iteration. The WHILE executes the statement_list as long as the search_condition is TRUE.
- Second, specify one or more statements that will execute between the DO and END WHILE keywords.
- Third, specify optional labels for the WHILE statement at the beginning and end of the loop construct.

Example 10:

```
DELIMITER $$
CREATE PROCEDURE my_proc_WHILE(IN n INT)
BEGIN
        SET @sum = 0;
        SET @x = 1;
WHILE @x<n DO
```

```
   IF mod(@x, 2) <> 0 THEN
                SET @sum = @sum + @x;
        END IF;
   SET @x = @x + 1;
END WHILE;
SELECT @sum;
END$$
DELIMITER ;
```

Odd numbers are numbers that cannot be divided exactly by 2. In the above procedure, a user passes a number through IN parameter and make a sum of odd numbers between 1 and that particular number.

6. Repeat

The REPEAT statement executes one or more statements until a search condition is true.

Here is the basic syntax of the REPEAT loop statement:

Syntax:

```
[begin_label:] REPEAT
   Statement
UNTIL search_condition
END REPEAT [end_label]
```

The REPEAT executes the statement until the search_condition evaluates to true.

The REPEAT checks the search_condition after the execution of statement, therefore, the statement always executes at least once. This is why the REPEAT is also known as a post-test loop.

The REPEAT statement can have labels at the beginning and at the end. These labels are optional.

This statement creates a stored procedure called RepeatDemo that uses the REPEAT statement to concatenate numbers from 1 to 9:

Example 11:

```
DELIMITER $$

CREATE PROCEDURE RepeatDemo()
BEGIN
   DECLARE counter INT DEFAULT 1;
   DECLARE result VARCHAR(100) DEFAULT '';

   REPEAT
      SET result = CONCAT(result,counter,',');
      SET counter = counter + 1;
   UNTIL counter >= 10
   END REPEAT;

   -- display result
   SELECT result;
END$$

DELIMITER ;
```

In this stored procedure:

- First, declare two variables counter and result and set their initial values to 1 and blank.
- The counter variable is used for counting from 1 to 9 in the loop. And the result variable is used for storing the concatenated string after each loop iteration.
- Second, append counter value to the result variable using the CONCAT() function until the counter is greater than or equal to 10.

7. Leave

The LEAVE statement exits the flow control that has a given label.

The following shows the basic syntax of the LEAVE statement:

Syntax:

```
LEAVE label;
```

LEAVE statement to exit a stored procedure

If the label is the outermost of the stored procedure or function block, LEAVE terminates the stored procedure or function.

Example 12:

```
DELIMITER $$
CREATE PROCEDURE CheckCredit(
inCustomerNumber int
)
sp: BEGIN
    DECLARE customerCount INT;
    SELECT COUNT(*)
    INTO customerCount
    FROM customers
    WHERE customerNumber = inCustomerNumber;
    IF customerCount = 0 THEN LEAVE sp;
    END IF;
END$$
DELIMITER ;
```

The LEAVE statement allows you to terminate a loop. The general syntax for the LEAVE statement when using in the LOOP, REPEAT and WHILE statements.

Using LEAVE with the REPEAT statement:

Syntax:

```
[label:] REPEAT
    IF condition THEN
        LEAVE [label];
    END IF;
    -- statements
UNTIL search_condition
END REPEAT [label];
```

Using LEAVE with the LOOP statement:

Syntax:

```
[label]: LOOP
    IF condition THEN
        LEAVE [label];
```

```
   END IF;
   -- statements
END LOOP [label];
```

Using LEAVE with the WHILE statement:

Syntax:

```
[label:] WHILE search_condition DO
   IF condition THEN
      LEAVE [label];
   END IF;
   -- statements
END WHILE [label];
```

Lab Task(s):

Exercise

---

1. Create a procedure **countEven** that will sum the number from 2 to given particular Number passes through IN parameter using any Loop.

2. Create a procedure **checkCustomer** that will take customerNumber through IN parameter and display Whether the customer exist in customer table or not.

3. Create a procedure **CountEmployees** that will use any conditional statement and count the no of employees based on these conditions

   - OFFICE_CODE > 3
   - OFFICE_CODE < 3
   - OFFICE_CODE = 3

   Call countEmployees(@count, 2);

The procedure takes two parameter, the office_code variable through IN parameter and return Count (number of employees) though out parameter.

4. Create a procedure EvenOdd that takes number through IN parameter then check and display whether the given no is Odd or Even.
5. Create 3 procedures (with LOOP, WHILE, REPEAT UNTIL) to print following output.

```
+------------+
| output     |
+------------+
| 1,2,3,4,5, |
+------------+
```

6. Create 3 procedures (with LOOP, WHILE, REPEAT UNTIL) to print following output.

```
+------------+
| output     |
+------------+
| 5,4,3,2,1, |
+------------+
```

7. Create 3 procedures (with LOOP, WHILE, REPEAT UNTIL) to print following output.

```
+--------------------+
| output             |
+--------------------+
| -5,-4,-3,-2,-1,0, |
+--------------------+
```