Iterative Statements

This lesson discusses the various statements supported by MySQL for repeated execution of commands. These are LOOP, WHILE, and REPEAT as well as ITERATE and LEAVE.

Iterative Statements

Iterative processing allows repeated execution of a set of statements. This is an important feature in database programming because we may need to loop through the rows returned by a query.

LOOP:

The most basic iterative statement is the **LOOP** statement. Any statements between the **LOOP** and **END LOOP** keywords are repeated until a condition for termination is met. The **LEAVE** statement is used to break the iterative processing.

> **[label]:** LOOP
>
> **statements;**
>
> IF **condition** THEN
>
> LEAVE **[label]**;

> END IF;
>
> ...
>
> END LOOP **[label]**;

The **LOOP** statement can start with an optional label which is used to refer to the loop. The **LEAVE** statement is used to break the execution. The **ITERATE** statement is used to ignore processing and start a new iteration of the loop.

WHILE:

WHILE statement provides an alternate way of iterative processing. It is better to understand because the terminating condition is clearly written between the **WHILE** and **DO** keywords as opposed to somewhere inside the **LOOP**. The **WHILE** statement is functionally similar to the **LOOP- LEAVE- END LOOP** construct as the terminating condition is checked before the execution of the loop begins. The syntax of while statement is:

> **[label]** WHILE **condition** DO
>
> **statements;**
>
> END WHILE **[label]**

The terminating condition is checked at the beginning of each iteration and if TRUE then the statements are executed. The process continues as long as the condition is TRUE. If the terminating condition is not true to begin with then the WHILE loop will not execute. Statements to be executed are provided between the **DO** and **END WHILE** keywords.

REPEAT:

Another iterative processing statement that is easier to read than the **LOOP** statement is **REPEAT**. It offers the same functionality as **WHILE** with one difference - the terminating condition is written at the end of the loop meaning that at least one iteration is performed before the condition is meaning that at least one iteration is performed before the condition is checked. **WHILE** loops are pre-test loops because they test the condition

before the statements are executed and the **REPEAT** loops are post-test loops as they test the condition after executing the loop statements. The termination condition is written after the **UNTIL** keyword.

```
[label:] REPEAT

statements;

UNTIL condition

END REPEAT [label]
```

LEAVE:

The **LEAVE** statement is used within the **LOOP** construct to terminate the execution of the loop. It can also be used to exit from a stored procedure. Basically the **LEAVE** statement that has a label associated with it, exits the flow control of that label. So in case of nested loops, we can break out of both loops with a single statement. The following syntax is used to exit a stored procedure:

```
CREATE PROCEDURE Procedure1() label:

BEGIN

statements;

IF condition THEN

LEAVE label;

END IF;

statements

END
```

> 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/55lesson.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 PrintMaleActors(
       OUT str  VARCHAR(255))
BEGIN
  DECLARE TotalRows INT DEFAULT 0;
  DECLARE CurrentRow INT;
  DECLARE fname VARCHAR (25);
  DECLARE lname VARCHAR (25);
  DECLARE gen VARCHAR (10);

  SET CurrentRow = 1;
  SET str =  '';

  SELECT COUNT(*) INTO TotalRows
  FROM Actors;

  Print_loop: LOOP
    IF CurrentRow > TotalRows THEN
      LEAVE Print_loop;
    END IF;

    SELECT Gender INTO gen
    FROM Actors
    WHERE Id = CurrentRow;

    IF gen NOT LIKE 'Male' THEN
      SET CurrentRow = CurrentRow + 1;
      ITERATE Print_loop;
    ELSE
      SELECT FirstName INTO fname
      FROM Actors
      WHERE Id = CurrentRow;

      SELECT SecondName INTO lname
      FROM Actors
      WHERE Id = CurrentRow;

      SET  str = CONCAT(str,fname,' ',lname,', ');
      SET CurrentRow = CurrentRow + 1;
    END IF;
  END LOOP Print_loop;
End **
DELIMITER ;
```

```
-- Query 2
CALL PrintMaleActors(@namestr);
SELECT @namestr AS MaleActors;

-- Query 3
DROP PROCEDURE PrintMaleActors;

DELIMITER **

CREATE PROCEDURE PrintMaleActors(
       OUT str  VARCHAR(255))
BEGIN

  DECLARE TotalRows INT DEFAULT 0;
  DECLARE CurrentRow INT;
  DECLARE fname VARCHAR (25);
  DECLARE lname VARCHAR (25);
  DECLARE gen VARCHAR (10);

  SET CurrentRow = 1;
  SET str =  '';

  SELECT COUNT(*) INTO TotalRows
  FROM Actors;

  Print_loop: WHILE CurrentRow < TotalRows DO
    SELECT Gender INTO gen
    FROM Actors
    WHERE Id = CurrentRow;

    IF gen LIKE 'Male' THEN
      SELECT FirstName INTO fname
      FROM Actors
      WHERE Id = CurrentRow;

      SELECT SecondName INTO lname
      FROM Actors
      WHERE Id = CurrentRow;

      SET  str = CONCAT(str,fname,' ',lname,', ');
    END IF;

    SET CurrentRow = CurrentRow + 1;
  END WHILE Print_loop;
End **
DELIMITER ;

-- Query 4
CALL PrintMaleActors(@namestr);
SELECT @namestr AS MaleActors;

-- Query 5
DROP PROCEDURE PrintMaleActors;
DELIMITER **
CREATE PROCEDURE PrintMaleActors(
       OUT str  VARCHAR(255))
BEGIN
  DECLARE TotalRows INT DEFAULT 0;
  DECLARE CurrentRow INT;
  DECLARE fname VARCHAR (25);
  DECLARE lname VARCHAR (25);
  DECLARE gen VARCHAR (10);

  SET CurrentRow = 1;
  SET str =  '';

  SELECT COUNT(*) INTO TotalRows
  FROM Actors;

  Print_loop: REPEAT
    SELECT Gender INTO gen
    FROM Actors
    WHERE Id = CurrentRow;

    IF gen LIKE 'Male' THEN
      SELECT FirstName INTO fname
      FROM Actors
      WHERE Id = CurrentRow;

      SELECT SecondName INTO lname
      FROM Actors
      WHERE Id = CurrentRow;

      SET  str = CONCAT(str,fname,' ',lname,', ');
    END IF;

    SET CurrentRow = CurrentRow + 1;
    UNTIL CurrentRow > TotalRows
  END REPEAT Print_loop;
End **
DELIMITER ;

-- Query 6
CALL PrintMaleActors(@namestr);
SELECT @namestr AS MaleActors;
```

Terminal

1. As an example we will create a stored procedure that prints the names of all male actors. We will use the **LOOP** statement to loop through the rows of the **Actors** table.

```
DELIMITER **
CREATE PROCEDURE PrintMaleActors(
        OUT str  VARCHAR(255))
BEGIN

  DECLARE TotalRows INT DEFAULT 0;
  DECLARE CurrentRow INT;
  DECLARE fname VARCHAR (25);
  DECLARE lname VARCHAR (25);
  DECLARE gen VARCHAR (10);
  SET CurrentRow = 1;

  SET str =  '';

  SELECT COUNT(*) INTO TotalRows
  FROM Actors;

  Print_loop: LOOP
    IF CurrentRow > TotalRows THEN
      LEAVE Print_loop;
    END IF;

  SELECT Gender INTO gen
  FROM Actors
  WHERE Id = CurrentRow;

  IF gen NOT LIKE 'Male' THEN
    SET CurrentRow = CurrentRow + 1;
    ITERATE Print_loop;
  ELSE
    SELECT FirstName INTO fname
    FROM Actors
    WHERE Id = CurrentRow;

    SELECT SecondName INTO lname
    FROM Actors
    WHERE Id = CurrentRow;

    SET  str = CONCAT(str,fname,' ',lname,', ');
    SET CurrentRow = CurrentRow + 1;
  END IF;
END LOOP Print_loop;

End **

DELIMITER ;
```

This stored procedure consists of an OUT parameter which is used to pass the string containing the names of the male actors. We have labelled our loop **Print_loop**. We first find the total number of rows in our table and store it in a variable **TotalRows**. The **IF** statement checks if the number of the row being currently examined is less than the **TotalRows** variable, otherwise the loop is terminated. The gender of each record is fetched in a local variable **gen**. The **ITERATE** statement restarts the execution at the start of the loop without executing the remaining statements. The statements in the **ELSE** block execute if the gender is male and add the name of the actor to the output string. Lastly the **CurrentRow** variable is incremented.

This is just a very cumbersome way of creating a list of names shown only as an example of using the **LOOP** statement in stored procedures. The same can be accomplished with a **SELECT** query by using **Gender** in the **WHERE** clause.

To test our procedure execute the following statements:

```
CALL PrintMaleActors(@namestr);
SELECT @namestr AS MaleActors;
```

2. The above example can be implemented using **WHILE** statement as follows:

```
DROP PROCEDURE PrintMaleActors;

DELIMITER **

CREATE PROCEDURE PrintMaleActors(
       OUT str  VARCHAR(255))
BEGIN

  DECLARE TotalRows INT DEFAULT 0;
  DECLARE CurrentRow INT;
  DECLARE fname VARCHAR (25);
  DECLARE lname VARCHAR (25);
  DECLARE gen VARCHAR (10);

  SET CurrentRow = 1;
  SET str =  '';

  SELECT COUNT(*) INTO TotalRows
  FROM Actors;

  Print_loop: WHILE CurrentRow < TotalRows DO
    SELECT Gender INTO gen
    FROM Actors
    WHERE Id = CurrentRow;
    IF gen LIKE 'Male' THEN
      SELECT FirstName INTO fname
      FROM Actors
      WHERE Id = CurrentRow;

      SELECT SecondName INTO lname
      FROM Actors
      WHERE Id = CurrentRow;

      SET  str = CONCAT(str,fname,' ',lname,', ');
    END IF;

    SET CurrentRow = CurrentRow + 1;
  END WHILE Print_loop;
End **

DELIMITER ;
```

The **WHILE** loop is labelled as **Print_loop**. **TotalRows** variable stores the total number of rows in our table. The termination condition is given between the **WHILE** and **DO** keywords and is checked before the loop begins execution. The gender of each record is fetched in a local variable **gen**. The statements following **IF** add the name of the actor to the output string if the gender is Male. The **CurrentRow** variable is incremented in the end and control moves to the beginning of the loop to check the termination condition.

To test our procedure execute the following statements:

```
CALL PrintMaleActors(@namestr);
SELECT @namestr AS MaleActors;
```

3. To demonstrate the **REPEAT** statement, we will use the same example used above. We can also observe the difference between **REPEAT** and **WHILE** statements.

```
DROP PROCEDURE PrintMaleActors;

DELIMITER **
```

```
CREATE PROCEDURE PrintMaleActors(
       OUT str  VARCHAR(255))
BEGIN

  DECLARE TotalRows INT DEFAULT 0;
  DECLARE CurrentRow INT;
  DECLARE fname VARCHAR (25);
  DECLARE lname VARCHAR (25);
  DECLARE gen VARCHAR (10);

  SET CurrentRow = 1;
  SET str =  '';

  SELECT COUNT(*) INTO TotalRows
  FROM Actors;

  Print_loop: REPEAT
    SELECT Gender INTO gen
    FROM Actors
    WHERE Id = CurrentRow;

    IF gen LIKE 'Male' THEN
      SELECT FirstName INTO fname
      FROM Actors
      WHERE Id = CurrentRow;

      SELECT SecondName INTO lname
      FROM Actors
      WHERE Id = CurrentRow;

      SET  str = CONCAT(str,fname,' ',lname,', ');
    END IF;

    SET CurrentRow = CurrentRow + 1;
    UNTIL CurrentRow > TotalRows
  END REPEAT Print_loop;

End **

DELIMITER ;
```

In the above example the loop is executed once before the termination condition is checked. The logic used in the loop is the same: if gender is male then add the name of the actor to the output string and increment the counter. The loop iterates until the terminating condition

CurrentRow > **TotalRows** is reached.

To test our procedure execute the following statements:

```
CALL PrintMaleActors(@namestr);
SELECT @namestr AS MaleActors;
```