Cursors

This lesson explores the concept of cursors which offer an efficient mechanism to process query results containing multiple rows.

We'll cover the following

Cursors

In the previous lesson on iterative processing we looped through a set of rows. Cursors is a kind of a loop which is used to traverse the rows returned by a query. Just as the cursor on the computer screen shows the current position, a database cursor also shows its current position in the result-set. Cursors can only be used in stored procedures, functions and triggers to loop through the results of a query one row at a time. Cursors have some properties which will be discussed next.

Cursors are **read-only** meaning they can only be used to view the result-set and not update it. They are **non-scrollable** meaning they can only show rows in the result-set in a sequential manner. It is not possible to view rows in a different order than the one returned by the query or to skip some rows to reach a specific row. MySQL cursors are **asensitive** meaning they point to the actual data in the table. There is another type of cursor which creates a temporary table for the result-set. This type is called insensitive cursors and these are not supported by MySQL. A potential drawback of working with the actual data in the table and not a temporary copy is that any changes made to the data from another connection can affect the results of the asensitive cursor.

To use a cursor, it is first declared using a **DECLARE** statement which

To use a cursor, it is first declared using a **DECLARE** statement which mentions the query with which the cursor is associated. It is necessary to declare the variables that will be used to manipulate the results returned by the query before declaring the cursor. Failure to do so will result in an error. The **OPEN** statement is used to initialize the cursor by fetching the rows resulting from the execution of the query. Next, to process each row of the result-set, the **FETCH** statement is used. This statement retrieves the row pointed to by the cursor and moves the pointer to the next row. As we are fetching rows one after the other, the cursor points to the next row in the result set. After fetching the last row, a condition is raised when the cursor cannot find the next row. To handle this situation we must define the **NOT FOUND** handler which sets a variable LastRowFetched to 1. This variable is checked in every iteration as the terminating condition of the loop. Lastly, the **CLOSE** statement is used to deactivate the cursor and release memory associated with it.

The following diagram illustrates the working of MySQL cursors:

Syntax

DECLARE **CursorName** CURSOR FOR

SELECTStatement;

DECLARE CONTINUE HANDLER FOR NOT FOUND

SET **LastRowFetched** = 1;

OPEN **cursor_name**;

FETCH **cursor_name** INTO **variables list**;

CLOSE **cursor_name**;

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/56lesson.sh** and wait for the MySQL prompt to start-up.

./DataJek/Lessons/56lesson.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 fName VARCHAR(25);
  DECLARE lName VARCHAR(25);
  DECLARE LastRowFetched INTEGER DEFAULT 0;
  DEClARE Cur_MaleActors CURSOR FOR
    SELECT FirstName, SecondName
    FROM Actors
    WHERE Gender = 'Male';
  DECLARE CONTINUE HANDLER FOR NOT FOUND
    SET LastRowFetched = 1;

  SET str =  '';
  OPEN Cur_MaleActors;

  Print_loop: LOOP
    FETCH Cur_MaleActors INTO fName, lName;
    IF LastRowFetched = 1 THEN
      LEAVE Print_loop;
    END IF;
    SET  str = CONCAT(str,fName,' ',lName,', ');
  END LOOP Print_loop;

  CLOSE Cur_MaleActors;
  SET LastRowFetched = 0;
END **
DELIMITER ;

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

Terminal

1. We will use the example from the previous lesson of a stored procedure for printing the names of the male actors. In this lesson, cursors will be used in the implementation. The **SELECT** query used to return the desired columns from the **Actors** table is:

   ```
   SELECT FirstName, SecondName FROM Actors WHERE Gender = 'Male';
   ```

   To manipulate the results we need two variables **fName** and **lName**. These variables need to be declared before we can declare the cursor. We also need to declare the variable used in the **NOT FOUND** Handler.

```
DECLARE fName VARCHAR(25);
DECLARE lName VARCHAR(25);
DECLARE LastRowFetched INTEGER DEFAULT 0;
```

After declaring variables, it is time to declare our cursor followed by the **NOT FOUND** handler as follows:

```
DEClARE Cur_MaleActors CURSOR FOR
  SELECT FirstName, SecondName
  FROM Actors
  WHERE Gender = 'Male';

DECLARE CONTINUE HANDLER FOR NOT FOUND
  SET LastRowFetched = 1;
```

Error handling is discussed at length in the next lesson where we will discuss the above handler statement in detail.

2. Next, to initialize the cursor we will use the **OPEN** statement which will load the results of the **SELECT** query in the cursor.

```
OPEN Cur_MaleActors;
```

3. Next step is to loop through the result set and concatenate the names into a comma separated list.

```
Print_loop: LOOP
  FETCH Cur_MaleActors
  INTO fName, lName;

  IF LastRowFetched = 1 THEN
    LEAVE Print_loop;
  END IF;

  SET  str = CONCAT(str,fName,' ',lName,', ');
END LOOP Print_loop;
```

The **LOOP** used above is much simpler than the one used in the previous lesson because cursors eliminate the need for manual manipulation of rows.

4. When the terminating condition is reached, control breaks from the loop. It is always a good practice to close a cursor when it is no longer in use. Another good practice is to reset the variable **LastRowFetched** back to 0. If this is not done then any subsequent or nested cursor loops can terminate prematurely.

```
CLOSE Cur_MaleActors;
SET LastRowFetched = 0;
```

5. Putting it all together, we can define our stored procedure as follows:

```
DELIMITER **

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

  DECLARE fName VARCHAR(25);
  DECLARE lName VARCHAR(25);
  DECLARE LastRowFetched INTEGER DEFAULT 0;

  DEClARE Cur_MaleActors CURSOR FOR
    SELECT FirstName, SecondName
    FROM Actors
    WHERE Gender = 'Male';

  DECLARE CONTINUE HANDLER FOR NOT FOUND
    SET LastRowFetched = 1;

  SET str =  '';

  OPEN Cur_MaleActors;

  Print_loop: LOOP
    FETCH Cur_MaleActors INTO fName, lName;

    IF LastRowFetched = 1 THEN
      LEAVE Print_loop;
    END IF;

    SET  str = CONCAT(str,fName,' ',lName,', ');
  END LOOP Print_loop;
  CLOSE Cur_MaleActors;

  SET LastRowFetched = 0;

END **
DELIMITER ;
```

To test the **PrintMaleActors** stored procedure use the following script:

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