UNIVERSIDAD
SAN SEBASTIAN
VOCACIÓN POR LA EXCELENCIA

ICIFF004 - Databases

19 - Iteration

Cristhian Aguilera (cristhian.aguilera@uss.cl)

Content

1. ICIFF004 - Databases
    1. Previous lectures
    2. The WHILE loop
    3. The REPEAT loop
    4. The LOOP loop
    5. Nested Loops
    6. Special cases and considerations
    7. Conclusions

Previous lectures

- Control statements in MySQL
  - IF-THEN-ELSE
  - ELSEIF
  - CASE

Example

```sql
DELIMITER //

CREATE PROCEDURE apply_discount(
    IN customer_id INT,
    IN purchase_amount DECIMAL(10,2)
)
BEGIN
  DECLARE discount DECIMAL(5,2);

  IF customer_id IN (SELECT id FROM loyal_customers) THEN
    SET discount = 0.10; -- 10% discount for loyal customers
  ELSE
    SET discount = 0.05; -- 5% discount for regular customers
  END IF;

  UPDATE orders
  SET total_amount = purchase_amount - (purchase_amount * discount)
  WHERE customer_id = customer_id;
END //

DELIMITER ;
```

Latest’s lecture exercises

1. Create a procedure to apply a discount into orders given the order_id.

- The discount must be applied to the TotalAmount
- The discount is 10% for TotalAmount > 500, 5% for TotalAmount > 200, and 0 otherwise

2. Alter the table Products to add a new column called StockStatus

- Create a procedure to check the stock status of a given product id
- There are three status: *Out of Stock* when StockQuantity is 0, *Low Stock* < 20, and *In Stock* > 20
- The procedure must update the StockStatus string of the given product ID accordingly

3. Create a function that can be used to determine the type of customer.

- The customer is *GOLD* if the total amount of purchases is over 1000
- *Silver* if > 500
- *Bronze* otherwise

What we will learn today

- Iteration in MySQL
  - WHILE
  - REPEAT
  - LOOP
  - LEAVE
  - ITERATE

The WHILE loop

- The WHILE statement is a looping construct that allows you to execute a block of code repeatedly based on a condition.
- The syntax of the WHILE statement is as follows:

```
WHILE condition DO
    statements;
END WHILE;
```

Example

```
CREATE PROCEDURE dowhile()
BEGIN
  DECLARE v1 INT DEFAULT 5;
  WHILE v1 > 0 DO
    ...
    SET v1 = v1 - 1;
  END WHILE;
END;
```

Example: Generating a sequence of numbers

In this example, the code generates a sequence of numbers from 1 to a specified limit.

```
DELIMITER //

CREATE PROCEDURE generate_sequence(IN top INT)
BEGIN
  DECLARE counter INT DEFAULT 1;

  WHILE counter ⩽ top DO
    INSERT INTO sequence_table (value) VALUES (counter);
    SET counter = counter + 1;
  END WHILE;
END //

DELIMITER ;
```

Example: Calculating the sum of squares

```sql
DELIMITER //

CREATE FUNCTION sum_of_squares(n INT)
RETURNS INT
DETERMINISTIC
BEGIN
  DECLARE sum INT DEFAULT 0;
  DECLARE i INT DEFAULT 1;

  WHILE i <= n DO
    SET sum = sum + (i * i);
    SET i = i + 1;
  END WHILE;

  RETURN sum;
END //

DELIMITER ;
```

The REPEAT loop

Similar to WHILE, but the condition is checked *after* each iteration.

```
REPEAT
    -- statements to execute repeatedly
UNTIL condition
END REPEAT;
```

Example: Finding the first even number in a table

A simple example to illustrate the REPEAT loop. The procedure searches for the first even number in a table and returns it.

```
DELIMITER //

CREATE PROCEDURE find_first_even(OUT even_num INT)
BEGIN
  DECLARE num INT DEFAULT 1;

  REPEAT
    SELECT value INTO num FROM numbers_table WHERE id = num;
    SET num = num + 1;
  UNTIL num % 2 = 0
  END REPEAT;

  SET even_num = num;
END //

DELIMITER ;
```

The LOOP loop

Provides more control with *LEAVE* (exit) and *ITERATE* (skip to next iteration). Similar to *break* and *continue* in other languages.

```
loop_label: LOOP
    -- statements to execute repeatedly
    IF condition THEN
        LEAVE loop_label;
    END IF;

    IF condition THEN
        ITERATE loop_label;
    END IF;
END LOOP loop_label;
```

Example: Processing data with a limit and skipping values

In this example, the procedure processes data up to a specified limit, skipping processing for certain values.

```sql
DELIMITER //
CREATE PROCEDURE process_data(IN data_limit INT)
BEGIN
  DECLARE counter INT DEFAULT 0;

  data_loop: LOOP
    SET counter = counter + 1;

    IF counter > data_limit THEN
      LEAVE data_loop;
    END IF;

    IF counter % 5 = 0 THEN
      ITERATE data_loop; -- Skip processing for multiples of 5
    END IF;
    -- Process data here...
  END LOOP data_loop;
END //

DELIMITER ;
```

Example: Searching for a value with an optional timeout

```
DELIMITER //
CREATE PROCEDURE search_with_timeout(
    IN target_value INT,
    IN timeout_seconds INT
)
BEGIN
  DECLARE start_time INT;
  DECLARE found_value BOOLEAN DEFAULT FALSE;
  SET start_time = UNIX_TIMESTAMP();

  search_loop: LOOP
    IF found_value THEN
      LEAVE search_loop;
    END IF;

    IF UNIX_TIMESTAMP() - start_time > timeout_seconds THEN
      LEAVE search_loop;
    END IF;
  END LOOP search_loop;
END //
DELIMITER ;
```

Within a stored function, *RETURN* can be used to exit the loop and return a value.

Nested Loops

Loops can be nested to handle multi-dimensional data or complex logic.

Example: Generating a simple text grid

Loops can be nested to generate, for example, a bidimensional search or data processing.

```sql
CREATE PROCEDURE print_grid(IN size INT)
BEGIN
  DECLARE i INT DEFAULT 1;
  DECLARE j INT;
  DECLARE grid_str VARCHAR(255) DEFAULT '';
  outer_loop: LOOP
    SET j = 1;
    inner_loop: LOOP
      SET grid_str = CONCAT(grid_str, '* ');
      SET j = j + 1;
      IF j > size THEN
        LEAVE inner_loop;
      END IF;
    END LOOP inner_loop;

    SET grid_str = CONCAT(grid_str, '\n');
    SET i = i + 1;
    IF i > size THEN
      LEAVE outer_loop;
    END IF;
  END LOOP outer_loop;
  SELECT grid_str;
END
```

Special cases and considerations

- **Infinite loops:** Ensure loop conditions eventually become *FALSE*.
- **Performance:** Minimize iterations and code complexity within loops.
  - Usually, set-based operations are more efficient than loops.

Let’s practice

- Convert the procedures of last lecture to use loops instead of ids. This will allow you to process all the data in the tables. (0.1 bonus point)

Conclusions

- Loops are essential for repetitive tasks and data processing in MySQL.
- *WHILE*, *REPEAT*, and *LOOP* offer different control mechanisms.
- *LEAVE* and *ITERATE* provide fine-grained control within loops.
- Understanding these constructs is crucial for writing efficient and robust stored programs.

Thanks for your attention!