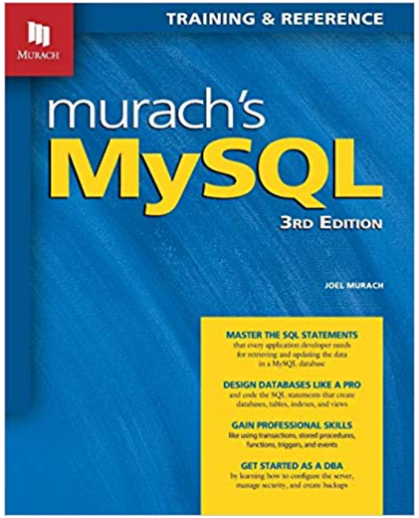
Extending SQL

Topic 4 Lesson 1
Extending SQL to include programming concepts

Adapted from Chapter 13, 15, 16



https://dev.mysql.com/doc/refman/8.0/en/trigger-syntax.html

https://dev.mysql.com/doc/refman/8.0/en/create-procedure.html

https://dev.mysql.com/doc/refman/8.0/en/stored-programs-defining.html

https://dev.mysql.com/doc/refman/8.0/en/sql-syntax-prepared-statements.html

SQL limitations

Compare the functionality of SQL to Python or Java. If we are going to build programming routines, what is missing in SQL?

We need a method for providing an environment and control flow to the program. We are missing the basic entities of a procedural language: variables, branching and iteration.

User Session Variable

- A session variable is a user-defined variable (not a server option) that starts with @, does not require declaration, can be used in any SQL query or statement, is not visible to other sessions, and exists until the end of the current session.
- Use set to assign a value to a variable

  SET @var = 1; or @var := 1;

  Variables not assigned a value has a default value of NULL
- Think of it as a **global variable for your current connection** – should only be used when working in the workbench when testing code

User variable values

- Data type for a variable is determined by the last assigned value
- User variables can be assigned a value from a limited set of data types: integer, decimal, floating-point, binary or non-binary string, or NULL value

Session or User Variable Limitations

- User variable defined by one client cannot be seen or used by other clients
- All variables for a given client session are automatically freed when that client exits
- A select expression is evaluated when it is sent to the client
    - **Do not expect a variable to be evaluated in a subquery**
    - All levels of the subquery will have the same value for the variable

User Variable Limitations (2)

- User variables are intended to provide data values to a SQL statement
  - They cannot provide a table name , a field name, or command literal to a query
    - EXCEPTION: Prepared statements
- User variables may be used in most SELECT SQL contexts where expressions are permitted
  - Exception: limit
- Do not assign a value to and read the value of the same variable within a single statement
  - The value is the initial value of the variable
    - Exception the SET command

Example: session variable

```
SET @row_number = 0;

SELECT @row_number := @row_number + 1, vendor_name
   FROM vendors;
```

Returns 2 columns where we generate a unique number from 1 to number of vendor names for the first column and a vendor_name for the second column.

Language extensions

Adds the following statements to your programming toolkit:

```
IF...ELSEIF...ELSE   END IF
CASE...WHEN...ELSE
WHILE...DO...LOOP
REPEAT...UNTIL...END REPEAT
DECLARE CURSOR FOR
DECLARE...HANDLER
```

Defining and using a local variable

Instead of a variable being known to the complete session, we can create local variables for local procedures or variables

The syntax for creating a local variable:

```
DECLARE variable_name data_type [DEFAULT literal_value];
```

The syntax for setting a variable to a literal value or an expression

```
SET variable_name = {literal_value|expression};
```

The syntax for setting a variable to a selected value

```
SELECT column_1[, column_2]...
   INTO variable_name_1[, variable_name_2]...
```

Syntax of an IF statement

```
IF boolean_expression THEN
   statement_1;
   [statement_2;]...
[ELSEIF boolean_expression
  THEN
   statement_1;
   [statement_2;]...]...
[ELSE
   statement_1;
   [statement_2;]...]
END IF;
```

Example

```
IF first_invoice_due_date
  < NOW() THEN
     SELECT 'Outstanding
  invoices are overdue!';
   ELSEIF
  first_invoice_due_date
         = NOW() THEN
     SELECT 'Outstanding
  invoices are due
  today!';
   ELSE
     SELECT 'No invoices
  are overdue.';
   END IF;
```

CASE statement

2 different versions:

Simple case statement evaluates an expression and each branch, considers a different value for the expression

Searched case statement supports multiple expressions. Each branch has its own expression to be evaluated.

Syntax for a Simple CASE statement

```
CASE expression
   WHEN expression_value_1
  THEN
      statement_1;
      [statement_2;]...
   [WHEN expression_value_2
  THEN
      statement_1;
      [statement_2;]...]...
   [ELSE
      statement_1;
      [statement_2;]...]
END CASE;
```

```
CASE terms_id_var
    WHEN 1 THEN
       SELECT ‘Net due 10
  days' AS Terms;
    WHEN 2 THEN
       SELECT 'Net due 20
  days' AS Terms;
    WHEN 3 THEN
       SELECT 'Net due 30
  days' AS Terms;
    ELSE
       SELECT 'Net due
  more than 30 days' AS
  Terms;
 END CASE;
```

Syntax for a Searched CASE statement

```
CASE
  WHEN expression_1 THEN
    statement_1;
    [statement_2;]...
  [WHEN expression_2 THEN
    statement_1;
    [statement_2;]...]...
  [ELSE
    statement_1;
    [statement_2;]...]
END CASE;
```

```
CASE
    WHEN terms_id_var = 1 THEN
      SELECT ‘Net due 10 days'
  AS Terms;
    WHEN terms_id_var = 2 THEN
      SELECT 'Net due 20 days'
  AS Terms;
    WHEN terms_id_var = 3 THEN
      SELECT 'Net due 30 days'
  AS Terms;
    ELSE
      SELECT 'Net due more than
  30 days' AS Terms;
  END CASE;
```

WHILE loop expression

SYNTAX

```
WHILE
  boolean_expression
  DO
   statement_1;
   [statement_2;]...
END WHILE;
```

EXAMPLE

```
DECLARE i INT;
DECLARE s VARCHAR(400)
           DEFAULT '';

WHILE i < 4 DO
    SET s = CONCAT(s, `i=`,
                i, ' | ');
    SET i = i + 1;
  END WHILE;
```

REPEAT Loop

EXAMPLE

```
SET i = 0;
REPEAT
   SET s = CONCAT(s,'i=',
  i,' | ');
   SET i = i + 1;
UNTIL i = 4
END REPEAT;
```

SYNTAX

```
REPEAT
      statement1….
      statement
      UNTIL  expression
      END  REPEAT;
```

Summary

We have added additional constructs to SQL in order to build programming database objects.