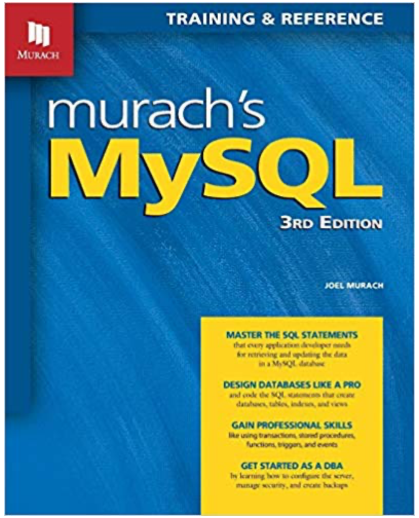
Database Programming Objects

Topic 4 Lesson 2
Procedures

Adapted from Chapter 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

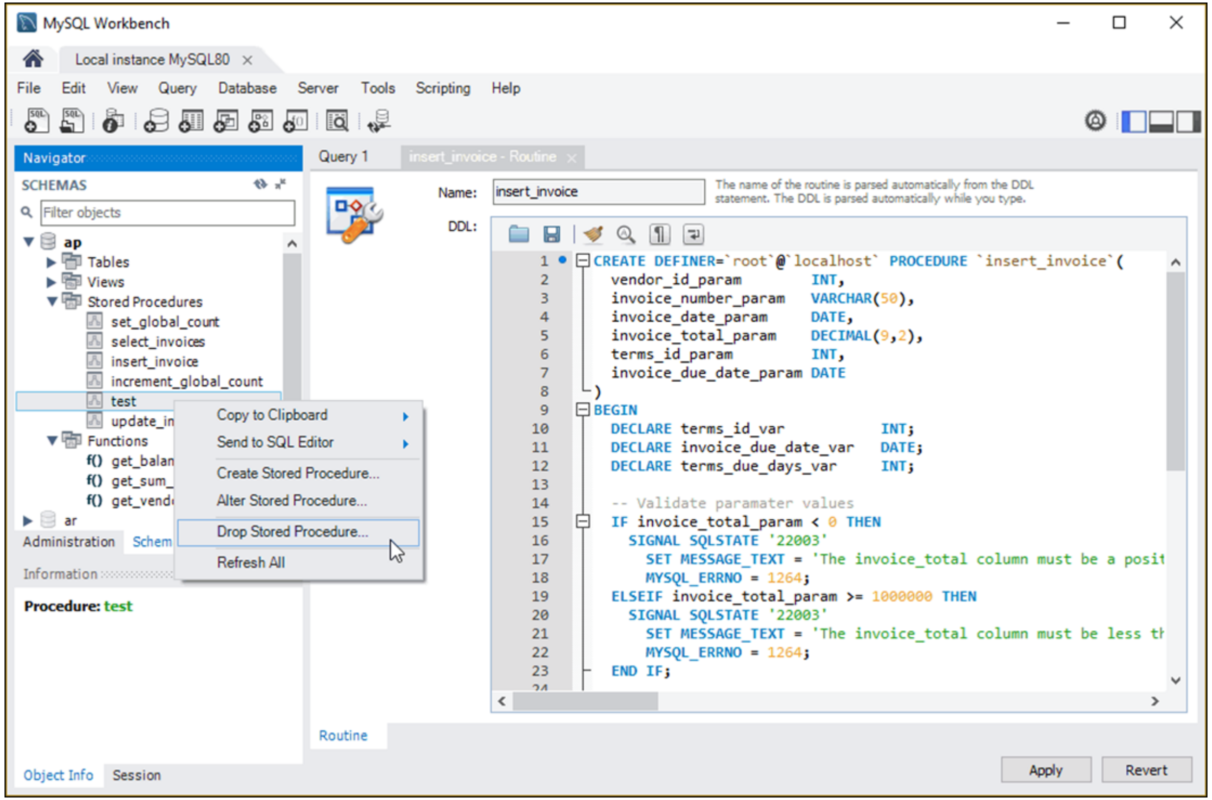
Stored database programming objects

- **Stored procedures**
  - An executable database object that contains a block of procedural SQL code
  - Use parameters to pass values to or from the procedure to the call program
  - Use the call program to execute a procedure
  - Can make changes to the database
  - Can return a two-dimensional result
- **Stored functions**
  - A user-defined function is an executable database object that contains a block of procedural SQL code
  - Functions can only return a scalar or single value
  - Function can only accept IN parameters
- **Trigger**
- **Event**

Locating database programming objects

Stored within the database

Users need executable permissions for the programming object



Procedure: test

Name: insert_invoice

The name of the routine is parsed automatically from the DDL statement. The DDL is parsed automatically while you type.

```sql
CREATE DEFINER=`root`@`localhost` PROCEDURE `insert_invoice`(
  vendor_id_param         INT,
  invoice_number_param    VARCHAR(50),
  invoice_date_param      DATE,
  invoice_total_param     DECIMAL(9,2),
  terms_id_param          INT,
  invoice_due_date_param DATE
)
BEGIN
  DECLARE terms_id_var          INT;
  DECLARE invoice_due_date_var   DATE;
  DECLARE terms_due_days_var     INT;

  -- Validate paramater values
  IF invoice_total_param < 0 THEN
    SIGNAL SQLSTATE '22003'
      SET MESSAGE_TEXT = 'The invoice_total column must be a posit
      MYSQL_ERRNO = 1264;
  ELSEIF invoice_total_param >= 1000000 THEN
    SIGNAL SQLSTATE '22003'
      SET MESSAGE_TEXT = 'The invoice_total column must be less th
      MYSQL_ERRNO = 1264;
  END IF;
```

Database Procedure

Stored procedure

Is executed using the CALL statement

Can return a two-dimensional result

Can pass value via the Arguments

- IN – argument used as input variable (default argument type)
- OUT – argument used as output variable
- INOUT – argument users as input and output variable

Example:

```
CREATE PROCEDURE n(IN val1, OUT val2, INOUT val3)
BEGIN
DECLARE local_var var_type;
END
```

Using local variables in a procedure

- Declare a variable in a procedure or function

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

- Setting a variable to a literal value or an expression

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

- Setting a variable to a selected value in a SELECT statement

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

Procedure example

```
USE ap;
 DROP PROCEDURE IF EXISTS test;

-- Change statement delimiter from semicolon to double front
  slash
DELIMITER //

CREATE PROCEDURE test()
BEGIN
  DECLARE sum_balance_due_var DECIMAL(9, 2);

  SELECT SUM(invoice_total - payment_total - credit_total)
  INTO sum_balance_due_var
  FROM invoices
  WHERE vendor_id = 95;
END //
DELIMITER ;
```

Question

What are the benefits to a database procedure?

Benefits to a stored procedure

- Operations are performed uniformly even for different programming languages
- Easier to maintain since the code is stored once in the database as opposed to duplicated in different applications
- Traffic is reduced between the client and the server since the stored procedure is executed on the server
- Security is enhanced – since clients can be granted fewer data base objects permission and still retrieve the data it needs
    - Application needs executable permissions for the stored procedure as opposed to read and write permission for the base tables

Procedure example

```
DELIMITER $$

CREATE PROCEDURE counter()
  BEGIN
  DECLARE x INT; -- example of DECLARE
  SET x = 1; -- EXAMPLE OF SET
   WHILE x <= 5 DO -- WHILE LOOP
         SET x = x + 1;
  END WHILE; -- CLOSE OF WHILE LOOP
  SELECT x; -- 6 -- THIS WILL PRINT THE VALUE OF THE VARIABLE
END $$

DELIMITER ;
```

Reassign the
delimiter from ; to $$ temporarily

Why do we need to do this?

DROP or CREATE a procedure

```
DROP PROCEDURE [IF EXISTS] procedure_name
```

A statement that creates a stored procedure

```
DELIMITER //

CREATE PROCEDURE clear_invoices_credit_total
(
  invoice_id_param  INT
)
BEGIN
  UPDATE invoices
  SET credit_total = 0
  WHERE invoice_id = invoice_id_param;
END//
```

Cursors

- A procedure can call the SELECT statement
- A SELECT statement can return a variable sized result
  - Multiple rows and multiple columns
- Procedure or application code uses a cursor to walk through each of the returned records one at a time

List of common errors:

https://dev.mysql.com/doc/refman/8.0/en/error-handling.html
https://dev.mysql.com/doc/refman/8.0/en/server-error-reference.html

MySQL cursor description:

https://dev.mysql.com/doc/refman/8.0/en/declare-handler.html

Cursor operations

Declare a cursor

```
DECLARE cursor_name CURSOR FOR select_statement;
```

Declare an error handler for when no rows are found in the cursor

```
DECLARE CONTINUE HANDLER FOR NOT FOUND handler_statement;
```

Open the cursor

```
OPEN cursor_name;
```

Get column values from the row and store them in a series of variables

```
FETCH cursor_name INTO variable1[, variable2][, variable3]...;
```

Close the cursor

```
CLOSE cursor_name;
```

Handler Declaration example

```
DECLARE CONTINUE
  HANDLER FOR ERROR
  error_handler;
```

Error is thrown when you try to read beyond the last record associated with a cursor

Define a handler for this error ***NOT FOUND*** to continue procedure control flow after reading all records

```
DECLARE CONTINUE HANDLER
  FOR NOT FOUND
      SET records = FALSE;
 -- records used to
  control loop for cursor
WHILE records DO
      -- process row
```

NOT FOUND: Shorthand for the class of SQLSTATE values that begin with '02'.
This is used to control what happens when a cursor reaches the end of a data set.

Procedure using a cursor

```
DELIMITER //

CREATE PROCEDURE apply_interest()
BEGIN
  DECLARE invoice_id_var    INT;
  DECLARE invoice_total_var DECIMAL(9,2);
  DECLARE row_not_found     TINYINT DEFAULT FALSE;
  DECLARE update_count      INT DEFAULT 0;

  DECLARE invoices_cursor CURSOR FOR
    SELECT invoice_id, invoice_total  FROM invoices
    WHERE invoice_total - payment_total - credit_total > 0;

  DECLARE CONTINUE HANDLER FOR NOT FOUND
    SET row_not_found = TRUE;

  OPEN invoices_cursor;

  WHILE row_not_found = FALSE DO
    FETCH invoices_cursor INTO invoice_id_var, invoice_total_var;
```

Procedure with cursor (continued)

```
IF invoice_total_var > 1000 THEN
      UPDATE invoices
      SET credit_total = credit_total
                       + (invoice_total * .1)
      WHERE invoice_id = invoice_id_var;
      SET update_count = update_count + 1;
    END IF;
  END WHILE;

  CLOSE invoices_cursor;

  SELECT CONCAT(update_count, ' row(s) updated.');
END//
```

Exit or Continue Handler

```
DECLARE {CONTINUE|EXIT}
  HANDLER
   FOR {

  mysql_error_code|
          SQLSTATE
  sqlstate_code|
          named_condition
         }
   handler_actions;
```

```
DECLARE CONTINUE HANDLER
  FOR NOT FOUND
     SET row_not_found
                   = TRUE;
```

An **EXIT** handler exits the current code block
A **Continue** handler does not exit the current code block

Example of a CONTINUE HANDLER

```
DELIMITER //

CREATE PROCEDURE test()
BEGIN
  DECLARE duplicate_entry_for_key TINYINT DEFAULT FALSE;

  DECLARE CONTINUE HANDLER FOR 1062
    SET duplicate_entry_for_key = TRUE;

  INSERT INTO general_ledger_accounts VALUES (130, 'Cash');

  IF duplicate_entry_for_key = TRUE THEN
    SELECT 'Row was not inserted - duplicate key encountered.'
        AS message;
  ELSE
    SELECT '1 row was inserted.' AS message;
  END IF;
END//
```

Example of an EXIT handler

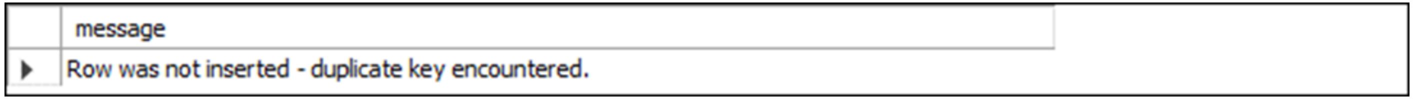
```
DELIMITER //

CREATE PROCEDURE test()
BEGIN
  DECLARE duplicate_entry_for_key TINYINT DEFAULT FALSE;
  BEGIN
    DECLARE EXIT HANDLER FOR 1062
      SET duplicate_entry_for_key = TRUE;

    INSERT INTO general_ledger_accounts VALUES (130, 'Cash');

    SELECT '1 row was inserted.' AS message;
  END;

  IF duplicate_entry_for_key = TRUE THEN
    SELECT 'Row was not inserted - duplicate key encountered.'
        AS message;
  END IF;
END//
```



```
message
Row was not inserted - duplicate key encountered.
```

Common Errors

Error code	SQLSTATE code	Occurs when a program…
1329	02000	attempts to fetch data from a row that doesn't exist.
1062	23000	attempts to store duplicate values in a column that has a unique constraint.
1048	23000	attempts to insert a NULL value into a column that doesn't accept NULL values.
1216	23000	attempts to add or update a child row but can't because of a foreign key constraint.
1217	23000	attempts to delete or update a parent row but can't because of a foreign key constraint.

Summary

Database procedures allow us to complete complex data processing on the database server as opposed to on the client system

Cursors allow applications as well as database procedures deal with the multisets that can be returned from SELECT statements. Cursors allow results to be processed row by row

Handler catches specific errors from called SQL statements. It is your responsibility to set up handlers for the errors you want to deal with.