MySQL Stored Routines

Sang Shin
http://www.javapassion.com
"Learning is fun!"

Topics

- Stored routines
- Stored procedures
  - IN, OUT, INOUT parameters
- Stored functions
  - Built-in functions
- Optional clauses
- Variables
- IF and CASE
- LOOP/WHILE/REPEAT
- CURSOR
- Handlers

Stored Routines

What is a Stored Routine?

- Captures a block of SQL statements in reusable and callable logic
- Associated with a specific database
- Two different kinds of stored routines
  - Stored procedures
  - Stored functions

Stored Procedure vs Stored Function

- Similarity
  - Both contain a block of SQL statements
- Differences
  - Stored function must produce a return value while stored procedures don't have to
  - Stored function cannot use SQL statements that return result sets
  - Stored function cannot use SQL statements that perform transactional commits or callbacks
  - Stored function call themselves recursively
  - Stored functions are called with SELECT while stored procedures are called with CALL

Advantages of Stored Routines

- Pre-compiled execution
- Reduced client/server traffic
- Efficient reuse of code and programming abstraction
- Enhanced security controls
  - Table specific security control can be hidden within the stored routines

Stored Procedures

Stored Procedure

- The body of the stored procedure can contain
  - SQL statements
  - Variable definitions
  - Conditional statements
  - Loops
  - Handlers
- BEGIN .. END markers are required when more than single statement makes the body
  - It is recommended even for a single statement for readability

Creating Stored Procedure

```
/* Delimiter is set to $$ so that you can use ;
 * inside body of the procedure.
 */
DELIMITER $$

/* Create a stored procedure */
CREATE PROCEDURE create_school_table()
BEGIN
   CREATE TABLE school_table (
          school_id INT NOT NULL,
          school_name VARCHAR(45) NOT NULL,
          PRIMARY KEY  (school_id)
   );
END $$

/* Change the delimiter back to ; */
DELIMITER ;
```

Input & Output Parameters of Stored Procedures

IN, OUT, INOUT

- IN parameters (default if not specified)
  - Serve as inputs to the procedure
- OUT parameters
  - Serve as outputs from the procedure
- INOUT parameters
  - Used both as input and outputs

IN Parameter

```
/* Definition of the procedure */
DELIMITER $$

CREATE PROCEDURE get_person(IN p_id SMALLINT)
BEGIN
   SELECT * FROM person
   WHERE person_id = p_id;
END $$

CREATE PROCEDURE get_person2(IN p_id SMALLINT, IN age INT)
BEGIN
   SELECT * FROM person
   WHERE person_id > p_id AND age > 10;
END $$

DELIMITER ;
/* End of procedure definition */
```

OUT Parameter

```
/* Definition of the procedure */
DELIMITER $$

CREATE PROCEDURE get_person_name(IN p_id SMALLINT,
                                 OUT f_name VARCHAR(45))
BEGIN
   SELECT first_name INTO f_name FROM person
   WHERE person_id = p_id;
END $$
/* End of the procedure definition */


/* Client then call the procedure as following */
CALL get_person_name(3, @myname);
SELECT @myname;
```

INOUT Parameter

```
/* Definition of the procedure */
DELIMITER $$

/* number is used both input and output */
CREATE PROCEDURE compute_square(INOUT number INT)
BEGIN
   SELECT number * number INTO number;
END $$
/* End of procedure definition */

/* Client then call the procedure as following */
SET @var=7;
CALL compute_square(@var);
SELECT @var;
```

Stored Functions

Stored Functions Examples

```
DELIMITER $$

CREATE FUNCTION compute_square_function(number INT)
RETURNS INT
BEGIN
   RETURN number * number;
END $$

CREATE FUNCTION compute_circle_area(radius INT)
RETURNS FLOAT
BEGIN
   RETURN PI() * radius * radius;
END $$

DELIMITER ;
```

Calling Stored Functions Examples

```
mysql>  SELECT compute_square_function(3);
+-------------------------------------+
| compute_square_function(3) |
+-------------------------------------+
|                                   9 |
+-------------------------------------+
1 row in set (0.09 sec)

mysql> SELECT compute_circle_area(3);
+-------------------------------------+
|         compute_circle_area(3) |
+-------------------------------------+
|              28.2743339538574 |
+-------------------------------------+
1 row in set (0.04 sec)
```

Optional Clauses for Stored Routines

Optional Clauses for Stored Routines

- DETERMINISTIC
  - The routine is deterministic - given the same input, it always products the same output
- LANGUAGE
  - The only possible value is SQL
- SQL SECURITY
  - Specifies which user's privileges should be considered when executing the routines
- COMMENT

Built-in Functions

Built-in functions

- Aggregate functions
  - AVG(), MAX(), MIN(), COUNT(), SUM()
- Mathematical functions
  - CEILING(), ABS(), PI(), RAND(), SQRT(), POWER(), ROUND()
- String functions
  - LENGTH(), CONCAT(), UPPER(), LOWER(), REPLACE(), SUBSTRING(), ASCII(), CHAR()

Variables

LOCAL VARIABLES

- Use DECLARE to declare variables that are local to a given routine
  - Optional DEFAULT
- Once defined, the local variables can be assigned values using either SET or SELECT .. INTO statements
- Accessing local variables
  - The local variables can be accessed by name from other statements within the same routine without @
  - They are accessed outside of the routine with @, however

VARIABLES Examples

```
DELIMITER $$

CREATE PROCEDURE declare_variables()
BEGIN
   DECLARE counter, return_value INT;
END $$


CREATE PROCEDURE compute_something_with_variables (IN number
INT)
BEGIN
   DECLARE my_value INT DEFAULT 9;

   SET @counter = number;
   SELECT @counter * my_value;
END $$
```

IF and CASE

IF, IF ELSE Examples

```
CREATE FUNCTION is_today_sunday()
RETURNS VARCHAR(255)
BEGIN
   DECLARE message VARCHAR(255) DEFAULT 'No';
   IF DAYOFWEEK(NOW()) = 1 THEN
      SET message = 'Yes';
   END IF;
   RETURN message;
END $$

CREATE FUNCTION what_is_today()
RETURNS VARCHAR(255)
BEGIN
   DECLARE message VARCHAR(255);
   IF DAYOFWEEK(NOW()) = 1 THEN
      SET message = 'Sunday';
   ELSEIF DAYOFWEEK(NOW()) = 2 THEN
      SET message = 'Monday';
   /* some code is omitted */
   END IF;
   RETURN message;
END $$
```

CASE Example

```
CREATE FUNCTION what_is_today_using_case()
RETURNS VARCHAR(255)
BEGIN
  DECLARE message VARCHAR(255);
  CASE DAYOFWEEK(NOW())
  WHEN 1 THEN
    SET message = 'Sunday';
  WHEN 2 THEN
    SET message = 'Monday';
  WHEN 3 THEN
    SET message = 'Tuesday';
  WHEN 4 THEN
    SET message = 'Wednesday';
  WHEN 5 THEN
    SET message = 'Thursday';
  WHEN 6 THEN
    SET message = 'Friday';
  WHEN 7 THEN
    SET message = 'Saturday';
  END CASE;
  RETURN message;
END $$
```

LOOP, WHILE, REPEAT

LOOP Example

```
CREATE FUNCTION factorial_loop(num INT UNSIGNED)
RETURNS INT
BEGIN
  DECLARE result INT DEFAULT 1;

  myloop: LOOP
    IF num > 0 THEN
      SET result = result * num;
      SET num = num - 1;
    ELSE
      LEAVE myloop;
    END IF;
  END LOOP myloop;

  RETURN result;
END $$
```

WHILE Example

```
CREATE FUNCTION factorial_while(num INT UNSIGNED)
RETURNS INT
BEGIN
  DECLARE result INT DEFAULT 1;

  myloop: WHILE num > 0 DO
    SET result = result * num;
    SET num = num - 1;
  END WHILE myloop;

  RETURN result;
END $$
```

REPEAT Example

```
CREATE FUNCTION factorial_repeat(num INT UNSIGNED)
RETURNS INT
BEGIN
  DECLARE result INT DEFAULT 1;

  myloop: REPEAT
    SET result = result * num;
    SET num = num - 1;
    UNTIL num <= 0
  END REPEAT myloop;

  RETURN result;
END $$
```

CURSOR

What is CURSOR?

- Used with LOOP/WHILE/REPEAT to process a collection of records (Result set) returned by SELECT
- CURSOR points to the current record in the collection
- Usage constraints
  - Forward-only
  - Read-only

How to Define CURSOR?

- CURSOR is initialized with DECALRE statement
- Each CURSOR is identified with a unique name and associated with a particular SELECT statement
  - *DECLARE <cursor-name> CURSOR FOR*
  - *SELECT first_name FROM person;*
- OPEN opens the cursor for reading
- FETCH reads contents of the current record into one or more variables and then advances the cursor to the next record
- CLOSE closes the cursor

CURSOR Example

```
CREATE PROCEDURE check_age_with_cursor()
BEGIN
  DECLARE f VARCHAR(255);
  DECLARE a INT;

  DECLARE c CURSOR FOR
    SELECT first_name, age FROM person;

  OPEN c;
  total: LOOP

    FETCH c INTO f, a;
    IF a > 60 THEN
      SELECT f AS FirstName, a AS Age, 'is Old' AS AgeCategory;
    ELSE
      SELECT f AS FirstName, a AS Age, 'is Young' AS AgeCategory;
    END IF;

  END LOOP total;
  CLOSE c;

END $$
```

Handlers

What is a Handler?

- Handler handles error conditions in stored procedures
- Steps for defining a handler
  - Declare error condition to be handled
  - Declare a handler for the named error condition
- Types of handlers
  - Exit handler - handles the error and then exit when an error occurs
  - Continue handler - handles the error and then continue when an error occurs

Error Conditions

- Error: 1329
  - SQLSTATE: 02000 (ER_SP_FETCH_NO_DATA)
  - Message: No data - zero rows fetched, selected, or processed
- Error: 1051
  - SQLSTATE: 42S02 (ER_BAD_TABLE_ERROR)
  - Message: Unknown table '%s'
- See MySQL error codes from
  - http://dev.mysql.com/doc/refman/5.5/en/error-messages-server.html

Exit Handler Example

```
CREATE PROCEDURE check_age_with_cursor_exit_handler()
BEGIN
  DECLARE f VARCHAR(255);
  DECLARE a INT;

  DECLARE c CURSOR FOR SELECT first_name, age FROM person;

  /* Declare error handler for no more records error condition */
  DECLARE EXIT HANDLER FOR 1329
  BEGIN
    SELECT 'We reached the end of the table!' AS message;
  END;

  OPEN c;
  total: LOOP

    FETCH c INTO f, a;
    /* Do something */

  END LOOP total;
  CLOSE c;

END $$
```

Continue Handler Example

```
/* An example procdure in which processing continues due to CONTINUE HANDLER */
CREATE PROCEDURE drop_table_continue_handler()
BEGIN

  /* Declare continue handler for non existent table error condition */
  DECLARE CONTINUE HANDLER FOR 1051
  BEGIN
    SELECT 'You are trying to drop a table that is non-existent!' AS message;
  END;

  SELECT 'Starting procedure' as message;
  /* You are dropping a table that does not exist.
   * This should cause an 1051 error condition.
   * Because there is CONTINUE HANDLER for 1051
   * error condition, the next statement should
   * be executed.
   */
  DROP TABLE non_existent_table;
  SELECT 'Ending procedure' as message;

END $$
```

Thank you!

Sang Shin
http://www.javapassion.com
“Learning is fun!”