Stored procedures and triggers

MySQL Stored Routines

- The routines are stored on server
- They belong to database
- They are based on standard SQL specification
- Three main components are
  - ✓ Procedure
  - ✓ Function
  - ✓ Trigger

MySQL Stored Procedures

- Parameter type
  - ✓ IN
  - ✓ OUT
  - ✓ INOUT
- The procedures may return one or more data sets as OUT parameters
- It is possible to use dynamic SQL
  - ✓ Dynamic SQL: construct dynamically statements as strings and then execute them

MySQL Stored Functions

- The MySQL function has only input parameters
- It must return one value of a given type
- It cannot be used with dynamic SQL
- It cannot return data sets

Example Procedure

```
mysql> delimiter //
mysql> CREATE PROCEDURE simpleproc
(OUT param1 INT)
-> BEGIN
-> SELECT COUNT(*) INTO param1 FROM t;
-> END;
-> //
Query OK, 0 rows affected (0.00 sec)
mysql> delimiter ;
mysql> CALL simpleproc(@a);
Query OK, 0 rows affected (0.00 sec)
mysql> DROP PROCEDURE simpleproc;
```

Example Procedure

```
mysql> delimiter //
mysql> CREATE PROCEDURE simpleproc
(OUT param1 INT)
-> BEGIN
-> SELECT COUNT(*) INTO param1 FROM t;
-> END;
-> //
Query OK, 0 rows affected
mysql> delimiter ;
mysql> CALL simpleproc(@a
Query OK, 0 rows affected
mysql> DROP PROCEDURE simpleproc;
```

```
mysql> SELECT @a;
+------+
| @a |
+------+
| 3  |
+------+
1 row in set (0.00 sec)
```

Example Function

```
CREATE FUNCTION hello (s CHAR(20))
RETURNS CHAR(50)
RETURN CONCAT('Hello, ',s,'!');
Query OK, 0 rows affected (0.00 sec)
mysql> SELECT hello('world');
+----------------+
| hello('world') |
+----------------+
| Hello, world! |
+----------------+
1 row in set (0.00 sec)
mysql> DROP FUNCTION hello;
```

Value passing in Procedure

```
CREATE PROCEDURE p (OUT ver_param
      VARCHAR(25), INOUT incr_param INT)
BEGIN
  # Set value of OUT parameter
  SELECT VERSION() INTO ver_param;
  # Increment value of INOUT parameter
  SET incr_param = incr_param + 1;
END;

mysql> SET @increment = 10;
mysql> CALL p(@version, @increment);
mysql> SELECT @version, @increment;
+------------+------------+
| @version   | @increment |
+------------+------------+
| 5.1.49     | 100        |
+------------+------------+
```

Flow Control

- BEGIN .. END blocks
- IF ... THEN ... ELSE ... END IF
- CASE ... THEN ... THEN ... ELSE ... END CASE
- WHILE ... END WHILE
- REPEAT ... UNTIL END REPEAT

- LOOP ... END LOOP
- ITERATE ***label***
  - ✓ ITERATE can appear only within LOOP, REPEAT, and WHILE statements. ITERATE means "start the loop again."
- LEAVE ***label***

LOOP Example

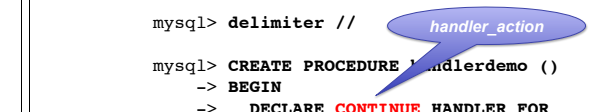
```
CREATE PROCEDURE doiterate(p1 INT)
BEGIN
  label1: LOOP
    SET p1 = p1 + 1;
    IF p1 < 10 THEN
      ITERATE label1;
    END IF;
    LEAVE label1;
  END LOOP label1;
  SET @x = p1;
END;
```

Exception Handlers

```
mysql> CREATE TABLE test.t (s1 INT, PRIMARY
KEY (s1));
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter //

mysql> CREATE PROCEDURE handlerdemo ()
    -> BEGIN
    ->   DECLARE CONTINUE HANDLER FOR
SQLSTATE '23000' SET @x2 = 1;
    ->   SET @x = 1;
    ->   INSERT INTO test.t VALUES (1);
    ->   SET @x = 2;
    ->   INSERT INTO test.t VALUES (1);
    ->   SET @x = 3;
    -> END;
    -> //
Query OK, 0 rows affected (0.00 sec)
```

Exception Handlers

```
mysql> CREATE TABLE test.t (s1 INT, PRIMARY
KEY (s1));
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter //

mysql> CREATE PROCEDURE handlerdemo ()
    -> BEGIN
    ->   DECLARE CONTINUE HANDLER FOR
SQLSTATE '23000' SET @x2 = 1;
    ->   SET @x = 1;
    ->   INSERT INTO test.t VALUES (1);
    ->   SET @x = 2;
    ->   INSERT INTO test.t VALUES (1);
    ->   SET @x = 3;
    -> END;
    -> //
Query OK, 0 rows affected (0.00 sec)
```



Exception Handlers

```
mysql> CREATE TABLE test (s1 INT, PRIMARY KEY
(s1));
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter //

mysql> CREATE PROCEDURE handlerdemo ()
    -> BEGIN
    ->   DECLARE CONTINUE HANDLER FOR
SQLSTATE '23000' SET @x2 = 1;
    ->   SET @x = 1;
    ->   INSERT INTO test VALUES (1);
    ->   SET @x = 2;
    ->   INSERT INTO test VALUES (1);
    ->   SET @x = 3;
    -> END;
    -> //
Query OK, 0 rows affected (0.00 sec)
```

condition_value: duplicate-key error

Exception Handlers

```
mysql> CALL handlerdemo()//
Query OK, 0 rows affected (0.00 sec)

mysql> mysql> select @x, @x2 //
+------+------+
| @x   | @x2  |
+------+------+
|    3 |    1 |
+------+------+
1 row in set (0.00 sec)
```

Cursor

```
CREATE PROCEDURE curdemo()
BEGIN
  DECLARE done INT DEFAULT 0;
  DECLARE a CHAR(16);
  DECLARE b, c DECIMAL(4,2);
  DECLARE cur1 CURSOR FOR SELECT id, data FROM test.data1;
  DECLARE cur2 CURSOR FOR SELECT data FROM test.data2;
  DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET done = 1;
  OPEN cur1;
  OPEN cur2;
  REPEAT
    FETCH cur1 INTO a, b;
    FETCH cur2 INTO c;
    IF NOT done THEN
      IF b < c THEN
        INSERT INTO test.data3 VALUES (a,b);
      ELSE
        INSERT INTO test.data3 VALUES (a,c);
      END IF;
    END IF;
  UNTIL done END REPEAT;
  CLOSE cur1;
  CLOSE cur2;
END//
```

No Data

MySQL Triggers

- A trigger is associated to table events (INSERT, UPDATE,DELETE)
- Its parameters depend on the event
- It does not return anything
- It is not possible to use it with dynamic SQL
- It cannot return data sets

```
DROP table s;
CREATE TABLE s (
staffid VARCHAR(5) PRIMARY KEY,
salary DECIMAL(6,2) NOT NULL,
work_done INTEGER NOT NULL,
bonus INTEGER
);
delimiter //
CREATE TRIGGER salary_bi
BEFORE INSERT ON s
FOR EACH ROW
BEGIN
  CASE
    WHEN new.work_done > 10  THEN SET new.bonus = 5000;
    WHEN new.work_done > 5   THEN SET new.bonus = 2500;
    WHEN new.work_done > 2   THEN SET new.bonus = 1000;
  ELSE
    SET new.bonus = 0;
  END CASE;
END//
delimiter ;
insert into s(staffid,salary,work_done) values ('s01', 100.0,
4);
```

Trigger_time: BEFORE, AFTER
Trigger_Event: INSERT, UPDATE, DELETE

Alias: OLD, NEW

MySQL Triggers

- ✓ The `OLD` and `NEW` keywords enable you to access columns in the rows affected by a trigger.
- ✓ In an `INSERT` trigger, only `NEW.col_name` can be used -- there is no old row. In a `DELETE` trigger, only `OLD.col_name` can be used -- there is no new row.
- ✓ In an `UPDATE` trigger, you can use `OLD.col_name` to refer to the columns of a row before it is updated and `NEW.col_name` to refer to the columns of the row after it is updated.
- ✓ A column named with `OLD` is read only. You can refer to it, but not modify it. A column named with `NEW` can be referred to if you have the `SELECT` privilege for it.
- ✓ In a `BEFORE` trigger, you can also change its value with `SET NEW.col_name = value` if you have the `UPDATE` privilege for it.

MySQL Triggers

- If a `BEFORE` trigger fails, the operation on the corresponding row is not performed.
- A `BEFORE` trigger is activated by the *attempt* to insert or modify the row, regardless of whether the attempt subsequently succeeds.
- An `AFTER` trigger is executed only if the `BEFORE` trigger (if any) and the row operation both execute successfully.
- An error during either a `BEFORE` or `AFTER` trigger results in failure of the entire statement that caused trigger invocation.