Navigating MySQL Stored Procedures & Functions and Triggers

Presented by:
Sheeri K. Cabral

At ODTUG Kaleidoscope 2010

Pythian
love your data

Who I Am

- MySQL DBA
- MySQL User Group
- First Oracle ACE Director for MySQL
- Lots of community stuff (videos, blog, podcast on hiatus)

Extended SQL Syntax

- No pl/sql
- Stored Routines
  - Stored procedures
  - Stored functions
- Views
- Triggers

Triggers

- Invoked automatically
- BEFORE, AFTER
- INSERT, UPDATE, DELETE
- 6 triggers per table

Data Changes w/out Triggering

- TRUNCATE
- DROP
- Foreign key cascading actions
  - Fixed when foreign keys in all storage engines
  - Still in the distance

Data Changes that Trigger

- REPLACE
  - Always INSERT, sometimes DELETE too
- INSERT...ON DUPLICATE KEY UPDATE
  - INSERT xor UPDATE
- LOAD DATA INFILE
  - insert

Creating a trigger

- TRIGGER privilege
  - global, db, tbl
  - different before MySQL 5.1.6

```
CREATE TRIGGER trg_name
[ BEFORE | AFTER ]
[ INSERT | UPDATE | DELETE ]
ON tbl_name
FOR EACH ROW ....
```

Conflicts

- Same trigger name

```
ERROR 1359 (HY000): Trigger already exists
```

- Same combination of:
  - BEFORE / AFTER
  - INSERT / UPDATE / DELETE

```
ERROR 1235 (42000): This version of MySQL
doesn't yet support 'multiple triggers with the
same action time and event for one table'
```

Sample Trigger

```
CREATE TRIGGER staff_update_date
BEFORE INSERT ON staff
FOR EACH ROW
SET NEW.create_date = NOW();
```

- NEW = alias for new row(s) inserted

NEW and OLD aliases

- NEW
  - `BEFORE INSERT`
  - `BEFORE UPDATE`
- OLD
  - `AFTER UPDATE`
  - `AFTER DELETE`
- NONE
  - `AFTER INSERT`
  - `BEFORE DELETE`

Dropping a Trigger

- `DROP TRIGGER trg_name;`
- `DROP TRIGGER IF EXISTS trg_name;`

Multiple SQL statements

- For triggers and others
- Will be using ;
  - So set DELIMITER first
- … `FOR EACH ROW BEGIN … END`

Multiple SQL Statements

```
DELIMITER |
CREATE TRIGGER before_staff_insert
BEFORE INSERT ON staff
FOR EACH ROW BEGIN
INSERT INTO staff_create_log (username,
created) VALUES (NEW.username, NOW());
SET NEW.last_update=NOW();
END |
DELIMITER ;
```

Changing a Trigger

- No ALTER TRIGGER

```
SELECT * FROM
INFORMATION_SCHEMA.TRIGGERS WHERE
TRIGGER_SCHEMA='db_name' AND
TRIGGER_NAME='trg_name'
```

```
SHOW CREATE TRIGGER trg_name;
```

Triggers on Special Tables

- Triggers are not supported on:
  - Views
  - Temporary tables

Trigger Runtime Behavior

- sql_mode
  - sql_mode of the creator
  - DROP, change sql_mode, re-CREATE
- charset and collation
  - Same as creator
  - DROP, change charset/collation, re-CREATE

Permissions

```
CREATE DEFINER=[ user@host | CURRENT_USER() ]
TRIGGER trg_name
[ BEFORE | AFTER ] [ INSERT|UPDATE|DELETE]
ON tbl_name FOR EACH ROW BEGIN … END;
```

- Requires SUPER privilege to set `user@host`

Finding Triggers

```
SELECT * FROM
INFORMATION_SCHEMA.TRIGGERS WHERE
TRIGGER_SCHEMA='db_name';
```

```
SHOW TRIGGERS;
```

```
SHOW TRIGGERS FROM 'db_name';
```

```
SHOW TRIGGERS LIKE 'trg_name';
```

Triggers and Replication

- `CREATE/DROP TRIGGER` not replicated
- Statement-based replication
  - Actions not saved to binary log
  - Put triggers on master and slave
- Row-based replication
  - All changes are saved to binary log
  - Triggers on master only
- Mixed acts like row-based

Triggers Cannot:

- Modify a table being used by the DML without NEW or OLD aliases
- Be defined on a `mysql` table
- Use `SELECT` without `INTO variable_name`
- Use `SHOW` commands
- Use `LOAD DATA/TABLE`
- Use `BACKUP/RESTORE DATABASE`

Triggers Cannot:

- Use prepared statement commands

`PREPARE, EXECUTE, DEALLOCATE PREPARE`

- Use `FLUSH` statements
- Invoke a UDF to call an external application
- Use `ALTER VIEW`
- Use `RETURN`

Triggers cannot cause COMMIT / ROLLBACK

`COMMIT`

`ROLLBACK`

`START TRANSACTION / BEGIN / BEGIN WORK`

`LOCK / UNLOCK TABLES`

`SET AUTOCOMMIT=1`

`TRUNCATE TABLE`

Most `ALTER / CREATE / DROP / RENAME` stmts

Stored Routines

- Performance
  - Cached **per connection**
- Stored procedure
  - Takes in 0 or more args
  - Outputs a result set
- Stored function
  - Takes in 0 or more args
  - Outputs a scalar value

Similar to Triggers

```
DELIMITER |

CREATE PROCEDURE store_offerings

(IN p_store_id TINYINT UNSIGNED, OUT
p_count INT UNSIGNED)

SELECT COUNT(*) INTO p_count

FROM inventory WHERE
store_id=p_store_id; |

DELIMITER ;
```

No Input/Output

```
DELIMITER |
CREATE PROCEDURE update_all_staff_time ()
UPDATE staff
SET last_update=NOW() WHERE 1=1; |
DELIMITER ;
```

Invoking a Stored Procedure

```
mysql> CALL store_offerings (1, @store_1);
Query OK, 0 rows affected (0.30 sec)

mysql> SELECT @store_1;
+-----------+
| @store_1 |
+-----------+
|      2270 |
+-----------+
1 row in set (0.00 sec)

CALL update_all_staff_time();
CALL update_all_staff_time;
```

Dropping a Stored Procedure

```
DROP PROCEDURE store_offerings;


DROP PROCEDURE IF EXISTS
store_offerings;
```

Multiple SQL statements

```
DELIMITER |
CREATE PROCEDURE update_all_staff_time ()
UPDATE staff
BEGIN
SET last_update=NOW() WHERE 1=1; |
END
DELIMITER ;
```

INOUT arguments

```
DELIMITER |
CREATE PROCEDURE increment_counter
(INOUT p_count INT UNSIGNED)
BEGIN SET p_count:=p_count+1;
END |
DELIMITER ;
```

Local variables

- `DECLARE var data_type` at body beginning

```
DELIMITER |
CREATE PROCEDURE pct_increase
(INOUT p_int INT, IN p_incr INT,
   OUT p_pct_incr DECIMAL (5,2))
BEGIN DECLARE p_int_new INT;
SET p_int_new := p_int + p_incr;
SET p_pct_incr := (p_int_new-p_int) /
                                p_int * 100;
SET p_int:=p_int_new;
END |
DELIMITER ;
```

Stored Procedure Runtime Behavior

- sql_mode
- charset
- collation
- Same as trigger – taken from definer

Security

- DEFINER and SQL SECURITY

```
CREATE
[ DEFINER = {user@host |
CURRENT_USER} ]
PROCEDURE p_name ( [param list] )
[SQL SECURITY  { DEFINER | INVOKER }
BEGIN … END
```

Other options (optional)

- `COMMENT`
- `[ NOT ] DETERMINISTIC`
- SQL usage
  - `MODIFIES SQL DATA`
  - `READS SQL DATA`
  - `CONTAINS SQL`
  - `NO SQL`
- ALTER PROCEDURE can change
  - `DEFINER, SQL SECURITY, COMMENT`

Full CREATE PROCEDURE Syntax

```
CREATE
[ DEFINER = { user@host | CURRENT_USER } ]
PROCEDURE p_name ( [ parameter_list ] )
[ option … ]
BEGIN … END
```

- Option is one or more of:

```
SQL SECURITY {DEFINER | INVOKER}
COMMENT 'comment string'
[NOT] DETERMINISTIC
[CONTAINS SQL | NO SQL | READS SQL DATA |
MODIFIES SQL DATA ]
```

Stored Function

- Very similar to stored procedure
- Output is scalar only
  - Only IN parameters, no INOUT or OUT
- Must use RETURNS clause
  - Defines what data type will be returned
- Must use RETURN statement at end
  - To actually return data

Example Stored Function

```
DELIMITER |

CREATE FUNCTION get_store_id (f_staff_id
TINYINT UNSIGNED)

RETURNS TINYINT UNSIGNED

BEGIN DECLARE f_store_id TINYINT UNSIGNED;

SELECT store_id INTO f_store_id FROM staff
WHERE staff_id=f_staff_id;

RETURN f_store_id;

END |

DELIMITER ;
```

Invoking a Stored Function

- No `CALL` like stored procedure
- Just as a regular function

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

Stored Routine Errors and Warnings

- As if statements were run on commandline
- No indication of which line failed
- No indication of which stored routine failed!

Conditions and Handlers

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

cond_stmt

- `SQLWARNING` - SQL state of warnings
- `NOT FOUND` – End of set for a cursor
- `SQLEXCEPTION` – not OK, SQLWARNING, NOT FOUND
- `mysql_error_num` – ie, 1265
- `SQLSTATE [VALUE] sqlstate` - 01000
- `condition_name`

Example

```
DELIMITER |
CREATE PROCEDURE pct_incr (INOUT p_int
INT, IN p_incr INT, OUT p_pct_incr DECIMAL
(5,2))
BEGIN DECLARE p_int_new INT UNSIGNED;
DECLARE CONTINUE HANDLER FOR 1265 SET
@warn_count:@warn_count+1;
SET p_int_new:=p_int+p_incr
SET p_pct_incr:=(p_int_new-p_int)/p_int *
100;
SET p_int:=p_int_new;
SELECT p_int, p_pct_incr, @warn_count;
END |
DELIMITER ;
```

Conditions

```
DECLARE condition_name CONDITION FOR
{mysql_err_code | SQLSTATE [VALUE]
sqlstate }
```

Instead of

```
DECLARE CONTINUE HANDLER FOR 1265 SET
@warn_count:=@warn_count+1;
```

DECLARE data_truncation CONDITION FOR 1265;

DECLARE CONTINUE HANDLER FOR data_truncation SET @warn_count:=@warn_count+1;

Stored Routine Flow Control

```
IF condition THEN stmt_list
[ ELSEIF condition THEN stmt_list ]
[ ELSE stmt_list ]
END IF
```

Stored Routine Flow Control

```
CASE WHEN condition THEN stmt_list
[ WHEN condition THEN stmt_list … ]
[ ELSE statement_list ]
END CASE
```

Loops

```
[label:] WHILE condition
DO statement_list
END WHILE [label]

[label:] REPEAT statement_list
UNTIL condition
END REPEAT [label]
```

Loops

```
[label:] LOOP
statement_list
END LOOP [label]

ITERATE label

LEAVE label
```

Cursors

```
DECLARE cursor_name CURSOR FOR
select_stmt;

OPEN cursor_name;

FETCH cursor_name INTO var_name [,
var_name]

…

CLOSE cursor_name;
```

Example cursor

```
DELIMITER |
CREATE PROCEDURE check_actors()
BEGIN DECLARE cur_actor SMALLINT UNSIGNED;
DECLARE film_count INT UNSIGNED;
DECLARE done,actor_count INT UNSIGNED DEFAULT 0;
DECLARE c_all_actors CURSOR FOR SELECT actor_id
FROM actor;
OPEN c_all_actors;
WHILE done=0 DO
FETCH c_all_actors INTO cur_actor;
SET actor_count:=actor_count+1;
SELECT COUNT(*) INTO film_count FROM film_actor
WHERE actor_id=cur_actor;
END WHILE;
CLOSE c_all_actors;
SELECT actor_count;
END |
DELIMITER ;
```

Questions, comments, etc?

- Views
- Post with links to play/download video, download slides, notes:

- http://www.technocation.org/node/621