Section 12 Stored Procedures and Scheduled Jobs

We often need to execute a number of sql statements in a particular order.
Store procedures allow us to store several statements together in one object that can be referenced.

Use project1
Create an AuditLog table:

```
CREATE TABLE AuditLog ( LogID INT NOT NULL AUTO_INCREMENT
                       ,Action VARCHAR(100) NULL
                       ,DateTimeStamp datetime NULL
                       ,PRIMARY KEY (LogID));
```

We are going to create a stored procedure that runs a few things and writes an audit log entry every time it goes.
Test and Insert Statement

```
  INSERT INTO AuditLog (Action , DateTimeStamp)
  VALUES ('Test1', Now());
```

```
mysql>   INSERT INTO AuditLog (Action , DateTimeStamp)
    ->   VALUES ('Test1', Now());
Query OK, 1 row affected (0.01 sec)

mysql> select * from AuditLog;
+-------+--------+---------------------+
| LogID | Action | DateTimeStamp       |
+-------+--------+---------------------+
|     1 | Test1  | 2016-08-05 21:06:03 |
+-------+--------+---------------------+
1 row in set (0.00 sec)
```

Let's create a stored procedure that can execute this command.
Write the stored proc in a file to make it easier to edit.
Note that the delimiter is what tells MySQL when a statement is complete. We are temporarily changing the delimiter to "//" so that we can put multiple statements inside the stored procedure. We then set the delimiter back to ";"

```
DELIMITER //
CREATE PROCEDURE MyStoredProc ()
BEGIN
  INSERT INTO AuditLog (Action , DateTimeStamp)
  VALUES ('Test1', Now());
END //
DELIMITER ;
```

```
StoredProc.sql   query.sql   quer
1  DELIMITER //
2  CREATE PROCEDURE MyStoredProc ()
3  BEGIN
4    INSERT INTO AuditLog (Action , DateTimeStamp)
5    VALUES ('Test1', Now());
6  END //
7  DELIMITER ;
8
```

```
mysql> source StoredProc.sql
Query OK, 0 rows affected (0.00 sec)
```

Now let's test our stored procedure;

```
mysql> select * from AuditLog;
+-------+--------+---------------------+
| LogID | Action | DateTimeStamp       |
+-------+--------+---------------------+
|     1 | Test1  | 2016-08-05 21:06:03 |
+-------+--------+---------------------+
1 row in set (0.00 sec)
```

SELECT * FROM AuditLog;
CALL MyStoredProc;
SELECT * FROM AuditLog;

```
mysql> CALL MyStoredProc;
Query OK, 1 row affected (0.01 sec)

mysql> select * from AuditLog;
+-------+--------+---------------------+
| LogID | Action | DateTimeStamp       |
+-------+--------+---------------------+
|     1 | Test1  | 2016-08-05 21:06:03 |
|     2 | Test1  | 2016-08-05 21:16:02 |
+-------+--------+---------------------+
2 rows in set (0.00 sec)
```

Let's make the stored procedure more dynamic by adding a parameter:

Start by dropping the old one:

```
DROP Procedure MyStoredProc;
```

```
mysql> DROP Procedure MyStoredProc;
Query OK, 0 rows affected (0.00 sec)
```

Then update StoredProc.sql:

```
DELIMITER //
CREATE PROCEDURE MyStoredProc (IN ActionDesc varchar(100))
BEGIN
  INSERT INTO AuditLog (Action , DateTimeStamp)
  VALUES (ActionDesc, Now());
END //
DELIMITER ;
```

```
StoredProc.sql   query.sql   query2.sql
1  DELIMITER //
2  CREATE PROCEDURE MyStoredProc (IN ActionDesc varchar(100))
3  BEGIN
4    INSERT INTO AuditLog (Action , DateTimeStamp)
5    VALUES (ActionDesc, Now());
6  END //
7  DELIMITER ;
8
```

```
mysql> source StoredProc.sql
Query OK, 0 rows affected (0.00 sec)
```

```
CALL MyStoredProc("I am the very model of a modern major general");
```

```
mysql> CALL MyStoredProc("I am the very modle of a modern major general");
Query OK, 1 row affected (0.01 sec)
```

```
mysql> select * from AuditLog;
+-------+---------------------------------------------+---------------------+
| LogID | Action                                      | DateTimeStamp       |
+-------+---------------------------------------------+---------------------+
|     1 | Test1                                       | 2016-08-05 21:06:03 |
|     2 | Test1                                       | 2016-08-05 21:16:02 |
|     3 | I am the very modle of a modern major general | 2016-08-05 21:29:43 |
+-------+---------------------------------------------+---------------------+
3 rows in set (0.00 sec)
```

The stored procedure can handle multiple actions.

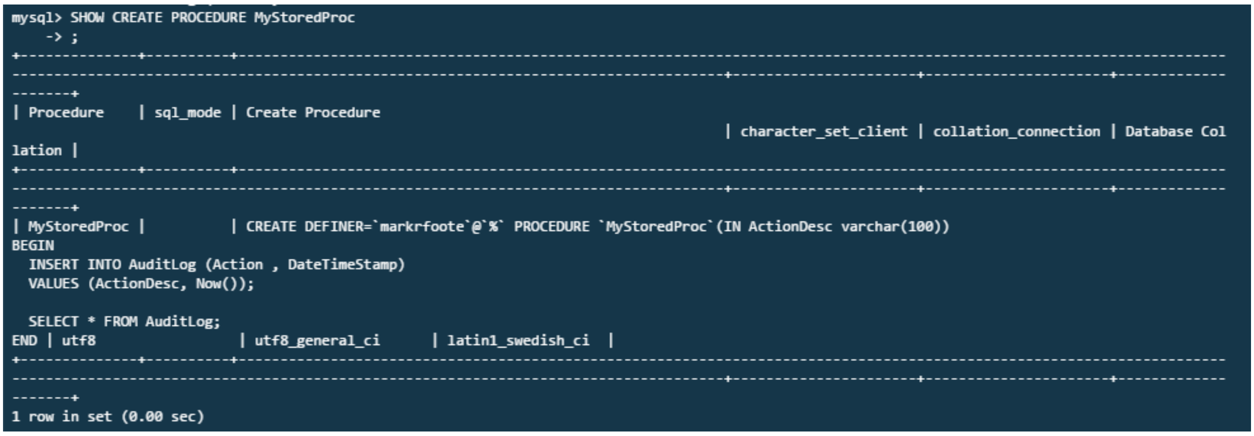
Let's DROP and Re-CREATE the stored proc with a SELECT * FROM AuditLog;

```
DELIMITER //
ALTER PROCEDURE MyStoredProc (IN ActionDesc varchar(100))
BEGIN
  INSERT INTO AuditLog (Action , DateTimeStamp)
  VALUES (ActionDesc, Now());

  SELECT * FROM AuditLog;
END //
DELIMITER ;
```

```
mysql> DROP Procedure MyStoredProc;
Query OK, 0 rows affected (0.00 sec)

mysql> source StoredProc.sql
Query OK, 0 rows affected (0.00 sec)

mysql> CALL MyStoredProc("I am the very modle of a modern major general");
+-------+---------------------------------------------+---------------------+
| LogID | Action                                      | DateTimeStamp       |
+-------+---------------------------------------------+---------------------+
|     1 | Test1                                       | 2016-08-05 21:06:03 |
|     2 | Test1                                       | 2016-08-05 21:16:02 |
|     3 | I am the very modle of a modern major general | 2016-08-05 21:29:43 |
|     4 | I am the very modle of a modern major general | 2016-08-05 21:53:07 |
+-------+---------------------------------------------+---------------------+
4 rows in set (0.01 sec)

Query OK, 0 rows affected (0.01 sec)
```

DROP Procedure MyStoredProc;
SOURCE StoredProc.sql
CALL MyStoredProc("I am the very model of a modern major general");

To check what's in an existing stored Procedure:
`SHOW CREATE PROCEDURE MyStoredProc`



```
mysql> SHOW CREATE PROCEDURE MyStoredProc
    -> ;
+--------------+----------+----------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------+----------------------+----------------------+-------------
-------+
| Procedure    | sql_mode | Create Procedure
                                                                                     | character_set_client | collation_connection | Database Col
lation |
+--------------+----------+----------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------+----------------------+----------------------+-------------
-------+
| MyStoredProc |          | CREATE DEFINER=`markrfoote`@`%` PROCEDURE `MyStoredProc`(IN ActionDesc varchar(100))
BEGIN
  INSERT INTO AuditLog (Action , DateTimeStamp)
  VALUES (ActionDesc, Now());

  SELECT * FROM AuditLog;
END | utf8                 | utf8_general_ci      | latin1_swedish_ci  |
+--------------+----------+----------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------+----------------------+----------------------+-------------
-------+
1 row in set (0.00 sec)
```

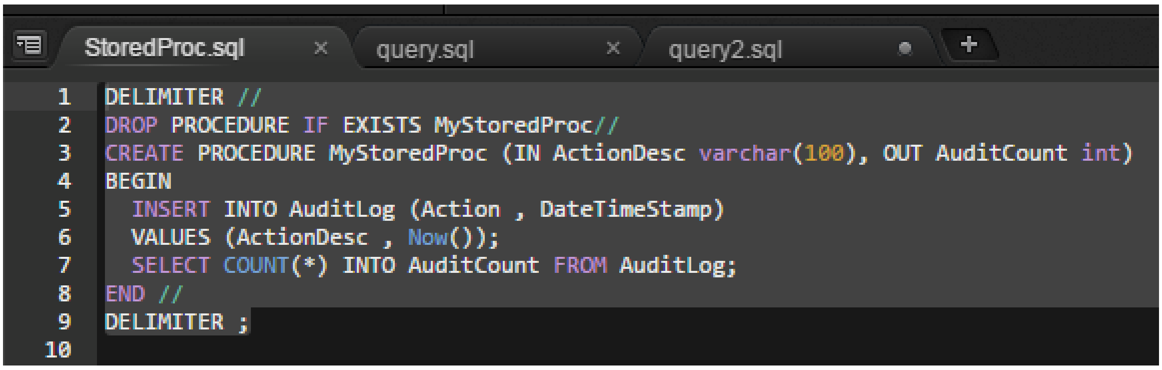
To make updating the stored procedure, add this line to the top of the file:

```
DROP PROCEDURE IF EXISTS MyStoredProc//
```

ALTER is an option for stored procedures but it has some limitations that make it frustrating to deal with. It's easier to drop and recreate.

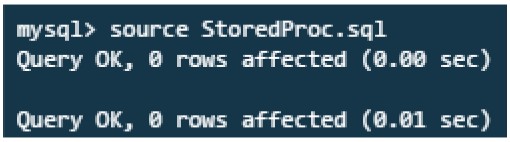
We are also going to add a return value in our stored procedure:

```
DELIMITER //
DROP PROCEDURE IF EXISTS MyStoredProc//
CREATE PROCEDURE MyStoredProc (IN ActionDesc varchar(100), OUT
AuditCount int)
BEGIN
  INSERT INTO AuditLog (Action , DateTimeStamp)
  VALUES (ActionDesc , Now());
  SELECT COUNT(*) INTO AuditCount FROM AuditLog;
END //
DELIMITER ;
```



```
DELIMITER //
DROP PROCEDURE IF EXISTS MyStoredProc//
CREATE PROCEDURE MyStoredProc (IN ActionDesc varchar(100), OUT AuditCount int)
BEGIN
  INSERT INTO AuditLog (Action , DateTimeStamp)
  VALUES (ActionDesc , Now());
  SELECT COUNT(*) INTO AuditCount FROM AuditLog;
END //
DELIMITER ;
```

```
source StoredProc.sql
```



```
mysql> source StoredProc.sql
Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.01 sec)
```

```
CALL MyStoredProc("Let It Go!",@MyReturnedValue);
```

```
SELECT @MyReturnedValue;

SELECT * FROM AuditLog;
```

```
mysql> CALL MyStoredProc("Let It Go!",@MyReturnedValue);
Query OK, 1 row affected (0.01 sec)

mysql> SELECT @MyReturnedValue;
+------------------+
| @MyReturnedValue |
+------------------+
|                5 |
+------------------+
1 row in set (0.00 sec)

mysql> SELECT * FROM AuditLog;
+-------+----------------------------------------------+---------------------+
| LogID | Action                                       | DateTimeStamp       |
+-------+----------------------------------------------+---------------------+
|     1 | Test1                                        | 2016-08-05 21:06:03 |
|     2 | Test1                                        | 2016-08-05 21:16:02 |
|     3 | I am the very modle of a modern major general | 2016-08-05 21:29:43 |
|     4 | I am the very modle of a modern major general | 2016-08-05 21:53:07 |
|     5 | Let It Go!                                   | 2016-08-05 22:20:30 |
+-------+----------------------------------------------+---------------------+
5 rows in set (0.00 sec)
```

You should notice that stored procedures act very similar to functions in several other programming languages.

Stored procedures are used in many ways. They are used to handle the SQL portion of an application. PHP for example would call the stored procedure and pass parameters. Other than stored procedure calls, no SQL actually needs to be in your PHP code.

Scheduled Jobs:

Scheduled jobs are an ideal way to use stored procedures. If you have a fast growing table that doesn't need more than 30 days worth of history, a stored procedure might be a good way to keep the table manageable. A scheduled job can kick of a stored procedure and any increment: ex. Every night do various cleanup items.

First we need the event_scheduler on:

Check by running `SHOW PROCESSLIST\G`

By default it won't be running:

```
mysql> SHOW PROCESSLIST\G
*************************** 1. row ***************************
     Id: 53
   User: markrfoote
   Host: localhost:52022
     db: project1
Command: Query
   Time: 0
  State: NULL
   Info: SHOW PROCESSLIST
*************************** 2. row ***************************
     Id: 62
   User: CRUD_application
   Host: localhost
     db: CRUD
Command: Sleep
   Time: 12271
  State:
   Info: NULL
```

```
SET GLOBAL event_scheduler = ON;
```

```
mysql> SHOW PROCESSLIST\G
*************************** 1. row ***************************
     Id: 53
   User: markrfoote
   Host: localhost:52022
     db: project1
Command: Query
   Time: 0
  State: NULL
   Info: SHOW PROCESSLIST
*************************** 2. row ***************************
     Id: 62
   User: CRUD_application
   Host: localhost
     db: CRUD
Command: Sleep
   Time: 12271
  State:
   Info: NULL
*************************** 3. row ***************************
     Id: 72
   User: event_scheduler
   Host: localhost
     db: NULL
Command: Daemon
   Time: 4
  State: Waiting on empty queue
   Info: NULL
3 rows in set (0.00 sec)
```

We can turn it back off using

SET GLOBAL event_scheduler = OFF;

```
mysql> SET GLOBAL event_scheduler = OFF;
Query OK, 0 rows affected (0.01 sec)

mysql> SHOW PROCESSLIST\G
*************************** 1. row ***************************
     Id: 53
   User: markrfoote
   Host: localhost:52022
     db: project1
Command: Query
   Time: 0
  State: NULL
   Info: SHOW PROCESSLIST
*************************** 2. row ***************************
     Id: 62
   User: CRUD_application
   Host: localhost
     db: CRUD
Command: Sleep
   Time: 12894
  State:
   Info: NULL
2 rows in set (0.00 sec)
```

For now, let's leave it on.

Now that the event_scheduler is on, we can now schedule stored procedures to by creating Events;

SHOW EVENTS;

```
mysql> SHOW EVENTS;
Empty set (0.09 sec)
```

Check out:http://dev.mysql.com/doc/refman/5.7/en/create-event.html

We can schedule them:`{YEAR | QUARTER | MONTH | DAY | HOUR | MINUTE | WEEK | SECOND | YEAR_MONTH | DAY_HOUR | DAY_MINUTE | DAY_SECOND | HOUR_MINUTE | HOUR_SECOND | MINUTE_SECOND}`

Let's create another sql file to work with: event.sql

```
CREATE EVENT MyScheduledStoredProcEvent
    ON SCHEDULE EVERY 1 MINUTE
    DO
      CALL MyStoredProc("I'm Running!",@MyReturnedValue);
```

```
event.sql   StoredProc.sql   query.sql
1 CREATE EVENT MyScheduledStoredProcEvent
2     ON SCHEDULE EVERY 1 MINUTE
3     DO
4       CALL MyStoredProc("I'm Running!",@MyReturnedValue);
5
```

In this example, we are setting the event to trigger every 1 minute

```
mysql> SOURCE event.sql
Query OK, 0 rows affected (0.02 sec)
```

```
SOURCE event.sql

SHOW EVENTS;
```

```
mysql> show events;
+----------+---------------------------+--------------+-----------+-----------+------------+----------------+----------------+---------------------+------+-----
----+------------+----------------------+----------------------+--------------------+
| Db       | Name                      | Definer      | Time zone | Type      | Execute at | Interval value | Interval field | Starts              | Ends | Stat
us  | Originator | character_set_client | collation_connection | Database Collation |
+----------+---------------------------+--------------+-----------+-----------+------------+----------------+----------------+---------------------+------+-----
----+------------+----------------------+----------------------+--------------------+
| project1 | MyScheduledStoredProcEvent | markrfoote@% | SYSTEM    | RECURRING | NULL       | 1              | MINUTE         | 2016-08-05 23:23:04 | NULL | ENAB
LED |          0 | utf8                 | utf8_general_ci      | latin1_swedish_ci  |
+----------+---------------------------+--------------+-----------+-----------+------------+----------------+----------------+---------------------+------+-----
----+------------+----------------------+----------------------+--------------------+
1 row in set (0.01 sec)
```

Watch the table grow:

```
mysql> SELECT * FROM AuditLog;
+-------+-----------------------------------------------+---------------------+
| LogID | Action                                        | DateTimeStamp       |
+-------+-----------------------------------------------+---------------------+
|     1 | Test1                                         | 2016-08-05 21:06:03 |
|     2 | Test1                                         | 2016-08-05 21:16:02 |
|     3 | I am the very modle of a modern major general | 2016-08-05 21:29:43 |
|     4 | I am the very modle of a modern major general | 2016-08-05 21:53:07 |
|     5 | Let It Go!                                    | 2016-08-05 22:20:30 |
|     6 | I'm Running!                                  | 2016-08-05 23:04:23 |
|     7 | I'm Running!                                  | 2016-08-05 23:23:04 |
|     8 | I'm Running!                                  | 2016-08-05 23:24:04 |
|     9 | I'm Running!                                  | 2016-08-05 23:25:04 |
+-------+-----------------------------------------------+---------------------+
9 rows in set (0.01 sec)
```

.

And before this AuditLog table get out of hand, let's drop the event:

```
mysql> DROP EVENT IF EXISTS MyScheduledStoredProcEvent;
Query OK, 0 rows affected (0.00 sec)
```

Challenge.

Create a table in the formcolletor database to keep track of ongoing statistics.

Have a column for the number of responses, average age, % male, % female respondents and the Candidate in the lead according to the last 7 days of polling.

Create a stored proc to look at the past 7 days and populate the new table with these details.

Create an event to run once every day and trigger the new stored proc.

Populate the PloiticalPoll table with enough data to allow you to test the functionality.

Feel free to change the interval so that you can test it quicker.

Hint 1:
SELECT SUM(if(GENDER = 'Male', 1, 0)) AS CountOfMaleRespondents
FROM PoliticalPoll;

Hint 2:

```
SELECT A.*
    ,D.CANDIDATE AS CandidateTrending
FROM PoliticalPoll A
   ,(SELECT C.CANDIDATE
     FROM (SELECT B.CANDIDATE
             ,COUNT(*)
         FROM PoliticalPoll B
         GROUP BY B.CANDIDATE
         ORDER BY 2 DESC
         LIMIT 1) C) D;
```

Solution:

Step 1: Create the Table

```
CREATE TABLE Stats ( StatsID INT NOT NULL AUTO_INCREMENT
                    ,Response7DayCount  INT NULL
                    ,RepondentAvgAge    DECIMAL(5,2) NULL
                    ,RepondentPctMale   DECIMAL(2,2) NULL
                    ,RepondentPctFemale DECIMAL(2,2) NULL
                    ,CandidateTrending  VARCHAR(20) NULL
                    ,DateTimeStamp datetime NULL
                    ,PRIMARY KEY (StatsID));
```

Step 2: Create the Select Statement

```
SELECT COUNT(*) AS Response7DayCount
--    ,NOW()
--    ,NOW() - INTERVAL 7 DAY
      ,ROUND(AVG(AGE),2)                                AS RepondentAvgAge
--    ,SUM(if(GENDER = 'Male', 1, 0))                   AS MaleCount
      ,ROUND(SUM(if(GENDER = 'Male', 1, 0))/COUNT(*),2)   AS RepondentPctMale
      ,ROUND(SUM(if(GENDER = 'Female', 1, 0))/COUNT(*),2) AS RepondentPctFemale
      ,D.CANDIDATE AS CandidateTrending
      ,NOW() AS DateTimeStamp
FROM PoliticalPoll A
     ,(SELECT C.CANDIDATE
       FROM (SELECT B.CANDIDATE
                   ,COUNT(*)
             FROM PoliticalPoll B
             GROUP BY B.CANDIDATE
             ORDER BY 2 DESC
             LIMIT 1) C) D
WHERE ENTRY_TimeStamp > NOW() - INTERVAL 7 DAY;
```

Step 3: Create the Insert Statement

```
INSERT INTO Stats ( Response7DayCount
                   ,RepondentAvgAge
                   ,RepondentPctMale
                   ,RepondentPctFemale
                   ,CandidateTrending
                   ,DateTimeStamp
                  )
SELECT COUNT(*) AS Response7DayCount
      ,ROUND(AVG(AGE),2)                                AS RepondentAvgAge
      ,ROUND(SUM(if(GENDER = 'Male', 1, 0))/COUNT(*),2)   AS RepondentPctMale
      ,ROUND(SUM(if(GENDER = 'Female', 1, 0))/COUNT(*),2) AS RepondentPctFemale
      ,D.CANDIDATE AS CandidateTrending
      ,NOW() AS DateTimeStamp
FROM PoliticalPoll A
     ,(SELECT C.CANDIDATE
       FROM (SELECT B.CANDIDATE
                   ,COUNT(*)
             FROM PoliticalPoll B
             GROUP BY B.CANDIDATE
             ORDER BY 2 DESC
             LIMIT 1) C) D
WHERE ENTRY_TimeStamp > NOW() - INTERVAL 7 DAY;
```

Step 4: Create the Store Procedure

```
DELIMITER //
CREATE PROCEDURE PopulateStats ()
BEGIN
  INSERT INTO Stats ( Response7DayCount
                     ,RepondentAvgAge
                     ,RepondentPctMale
                     ,RepondentPctFemale
                     ,CandidateTrending
                     ,DateTimeStamp
                    )
  SELECT COUNT(*) AS Response7DayCount
        ,ROUND(AVG(AGE),2)                                  AS RepondentAvgAge
        ,ROUND(SUM(if(GENDER = 'Male', 1, 0))/COUNT(*),2)   AS RepondentPctMale
        ,ROUND(SUM(if(GENDER = 'Female', 1, 0))/COUNT(*),2) AS RepondentPctFemale
        ,D.CANDIDATE AS CandidateTrending
        ,NOW() AS DateTimeStamp
  FROM PoliticalPoll A
       ,(SELECT C.CANDIDATE
         FROM (SELECT B.CANDIDATE
                     ,COUNT(*)
               FROM PoliticalPoll B
               GROUP BY B.CANDIDATE
               ORDER BY 2 DESC
               LIMIT 1) C) D
  WHERE ENTRY_TimeStamp > NOW() - INTERVAL 7 DAY;
END //
DELIMITER ;
```

Test It: `CALL PopulateStats;`
`SELECT * FROM Stats;`

Step 5: Create the Event

```
CREATE EVENT Populate7DayStats
    ON SCHEDULE EVERY 1 MINUTE
    DO
      CALL PopulateStats;

SHOW EVENTS;
SELECT * FROM Stats;
```

Step 6: Test it

Add Data and watch the rows get updated

Step 7:Update the schedule and Purge test data

```
DROP EVENT IF EXISTS Populate7DayStats;
CREATE EVENT Populate7DayStats
    ON SCHEDULE EVERY 1 DAY
    DO
      CALL PopulateStats;

SHOW EVENTS;

DELETE FROM Stats;

SELECT * FROM Stats;
```