MySQL 5.0 Triggers

MySQL 5.0 New Features Series – Part 2

A MySQL® Technical White Paper

Peter Gulutzan

March, 2005

Table of Contents

Introduction

This book is for long-time MySQL users who want to know "what's new" in version 5. The short answer is "stored procedures, triggers, views, information_schema". The long answer is the “MySQL 5.0 New Features” series, and this book is the second in the series.

What I'm hoping to do is make this look like a hands-on session where you, as if you're working it out yourself on your keyboard, can walk through sample problems.

To do this, I'll go through each little item, building up slowly. By the end, I'll be showing larger routines that do something useful, something that you might have thought was tough.

Conventions and Styles

Whenever I want to show actual code, such as something that comes directly from the screen of my mysql client program, I switch to a Courier font, which looks different from the regular text font. For example:

```
mysql> DROP FUNCTION f;
Query OK, 0 rows affected (0.00 sec)
```

When the example is large and I want to draw attention to a particular line or phrase, I highlight it with a double underline and a small arrow on the right of the page. For example:

```
mysql> CREATE PROCEDURE p ()
    -> BEGIN
    ->   /* This procedure does nothing */              <--
    -> END;//
Query OK, 0 rows affected (0.00 sec)
```

Sometimes I will leave out the “mysql>” and “->” prompts so that you can cut the examples and paste them into your copy of the mysql client program. (If you aren't reading the text of this book in a machine-readable form, try looking for the script on the mysql.com web site.)

I tested all the examples with the publicly-available alpha version of MySQL 5.0.3 on Linux, SUSE 9.2. By the time you read this, the version number will be higher and the available operating systems will include Windows, Sparc, and HP-UX. So I'm confident that you'll be able to run every example on your computer. But if not, well, as an experienced MySQL user you know that help and support is always available.

Why Triggers

We are including support for triggers in MySQL 5.0 for these reasons:

- Users of earlier MySQL versions kept on telling us they wanted triggers.
- We have a commitment to support all ANSI-standard features.
- You can use them to check for, and prevent, bad data entering the database.
- You can change or negate the INSERT, UPDATE, and DELETE statements.
- You can monitor data-change activity throughout a session.

I assume that you have read the first book in the “MySQL New Features” series already. In that book, “MySQL Stored Procedures”, you (I hope) saw how MySQL supports stored procedures and functions. That's important knowledge, because you can use the same statements in triggers as you can use in functions. Specifically:

• Compound statements (BEGIN / END) are legal.
• Flow-of-control statements (IF, CASE, WHILE, LOOP, WHILE, REPEAT, LEAVE, ITERATE) are legal.
• Variable declaration (DECLARE) and assignment (SET) are legal.
• Condition declarations are legal.
• Handler declarations are legal.

But remember that functions are subject to severe limitations: you cannot access tables from within a function. So these statements are illegal inside a function:

```
ALTER 'CACHE INDEX' CALL COMMIT CREATE DELETE
DROP 'FLUSH PRIVILEGES' GRANT INSERT KILL
LOCK OPTIMIZE REPAIR REPLACE REVOKE
ROLLBACK SAVEPOINT 'SELECT FROM table'
'SET system variable' 'SET TRANSACTION'
SHOW 'START TRANSACTION' TRUNCATE UPDATE
```

Precisely the same limitation applies for triggers.

Triggers are very new. There are bugs. Therefore I give the same warning that I gave for stored procedures. Do not try triggers with a database that has important data in it. Instead, create a new database for testing purposes, and make sure this test database is the default, whenever you are creating or using tables with triggers.

Syntax

1. Syntax: Name

```
CREATE TRIGGER <trigger name>                                <--
  { BEFORE | AFTER }
  { INSERT | UPDATE | DELETE }
  ON <table name>
  FOR EACH ROW
  <triggered SQL statement>
```

Triggers must have names, up to 64 characters long, possibly enclosed in backtick `delimiters`. In other words, they're much like names of other objects in MySQL.

I use a convention: trigger names are table_name + '_' + trigger_type_abbreviation. So for table t26, trigger time BEFORE and trigger event UPDATE (see points (2) and (3) below), the trigger name would be t26_bu.

2. Syntax: Time

```
CREATE TRIGGER <trigger name>
  { BEFORE | AFTER }                                    <--
  { INSERT | UPDATE | DELETE }
  ON <table name>
  FOR EACH ROW
  <triggered SQL statement>
```

Triggers have an action time: they can be activated before an event or after an event.

3. Syntax: Event

```
CREATE TRIGGER <trigger name>
  { BEFORE | AFTER }
  { INSERT | UPDATE | DELETE }                           <--
  ON <table name>
  FOR EACH ROW
  <triggered SQL statement>
```

Triggers also have an event: they can be activated during insert, or during update, or during delete.

4. Syntax: Table

```
CREATE TRIGGER <trigger name>
  { BEFORE | AFTER }
  { INSERT | UPDATE | DELETE }
  ON <table name>                                        <--
  FOR EACH ROW
  <triggered SQL statement>
```

Triggers have a subject table: it is inserts or updates or deletes on this subject table that cause trigger activation. I can't make two triggers for the same table and the same event.

5. Syntax: Granularity

```
CREATE TRIGGER <trigger name>
  { BEFORE | AFTER }
  { INSERT | UPDATE | DELETE }
  ON <table name>
  FOR EACH ROW                                           <--
  <triggered SQL statement>
```

Triggers have a granularity: the FOR EACH ROW clause says that trigger activation will occur for the rows of the table, not for the table as a whole.

6. Syntax: Statement

```
CREATE TRIGGER <trigger name>
  { BEFORE | AFTER }
  { INSERT | UPDATE | DELETE }
  ON <table name>
  FOR EACH ROW
  <triggered SQL statement>              <--
```

Triggers have a triggered SQL statement: the statement can be any statement, including a compound statement, but triggers have all the same limitations as functions.

Privileges

You'll need the SUPER privilege for CREATE TRIGGER. If you're root, you have it. This is a non-standard requirement, and I expect that we'll change it soon.

So in the next version of MySQL, you'll probably see that there is a new privilege named the CREATE TRIGGER privilege. Then you'll be able to grant with this:

```
GRANT CREATE TRIGGER ON <table name> TO <user list>;
```

And you'll be able to revoke with this:

```
REVOKE CREATE TRIGGER ON <table name> FROM <user list>;
```

Referring to OLD and NEW columns

Within a triggered SQL statement, you can refer to any column of the table. But you can't just say “column_name” -- that would sometimes be ambiguous, because there might be a new column name (what you're changing to) as well as an old column name (what you're changing from). So you have to say “NEW . column_name” or “OLD . column_name”. The technical term for the objects you're referring to with NEW | OLD . column_name is “transition variables”.

With INSERT, only NEW is legal. With DELETE, only OLD is legal. With UPDATE, both NEW and OLD are legal. Here is an example of an UPDATE trigger with both:

```
CREATE TRIGGER t21_au
BEFORE UPDATE ON t22
FOR EACH ROW
BEGIN
  SET @old = OLD . s1;
  SET @new = NEW.s1;
END;//
```

Now, if t21 has one row containing 55 in column s1, and I say “UPDATE t21 SET s1 = s1 + 1”, then after the update is finished, @old will be 55 and @new will be 56.

Example of CREATE and INSERT

CREATE table with trigger

For this and all examples, I assume that the delimiter has already been set, with
`DELIMITER //`

```
CREATE TABLE t22 (s1 INTEGER)//

CREATE TRIGGER t22_bi
BEFORE INSERT ON t22
FOR EACH ROW
BEGIN
  SET @x = 'Trigger was activated!';
  SET NEW.s1 = 55;
END;//
```

I begin by creating a test table named t22. Then I create a trigger on table t22, so that before I insert any row into the table, I set a flag that the trigger was activated, and I change the new value of a column in the table to fifty-five.

INSERT on table with a trigger

```
mysql> INSERT INTO t22 VALUES (1)//
```

Let's see what happens if I insert a row into table t2, the trigger's subject table.

The INSERT statement is utterly ordinary. I do not need to have “privileges on the trigger”. I do not even need to know that the trigger exists.

```
mysql> SELECT @x, t22.* FROM t22//
+------------------------+------+
| @x                     | s1   |
+------------------------+------+
| Trigger was activated! |   55 |
+------------------------+------+
1 row in set (0.00 sec)
```

Here we see what happens as a result of the INSERT. The x flag is set, as expected. And the value that actually gets inserted isn't what I said to INSERT in the INSERT statement. Instead, it's what I said to INSERT in the trigger.

Example of a “check” constraint

What's a “check” constraint

In standard SQL, one can say “CHECK (condition)” as part of a CREATE TABLE statement, for example

```
CREATE TABLE t25
  (s1 INT, s2 CHAR(5), PRIMARY KEY (s1),
  CHECK (LEFT(s2,1)='A'))
  ENGINE=INNODB;
```

The CHECK means “insert and update are illegal unless the leftmost character of column s2 is 'A'”. MySQL doesn't support that. MySQL does support the CHECK clause for views, which is what I'd recommend. But just in case you insist on putting the equivalent of a CHECK clause on a base table, here's how you can do it with triggers.

```
CREATE TABLE t25
(s1 INT, s2 CHAR(5),
PRIMARY KEY (s1))
ENGINE=INNODB//

CREATE TRIGGER t25_bi
BEFORE INSERT ON t25
FOR EACH ROW
IF LEFT(NEW.s2,1)<>'A' THEN SET NEW.s1=0; END IF;//

CREATE TRIGGER t25_bu
BEFORE UPDATE ON t25
FOR EACH ROW
IF LEFT(NEW.s2,1)<>'A' THEN SET NEW.s1=0; END IF;//
```

I only need BEFORE INSERT and BEFORE UPDATE triggers. DELETE triggers won't matter, and AFTER triggers can't change NEW transition variables.

I have to prime the pump by inserting a row with s1=0. After that, any attempt to add or change so that LEFT(s2,1) <> 'A' will fail. Try it with:

```
INSERT INTO t25 VALUES (0,'a') /* priming the pump */   //
INSERT INTO t25 VALUES (5,'b') /* gets error '23000' */ //
```

Don't Believe The Old MySQL Manual

I am going to warn you that you cannot believe everything that you read in the MySQL manual in days of yore. We have removed false statements about triggers, but there are still old versions of the manual on the web. For example, here's a German URL that's still wrong at time of writing: http://dev.mysql.com/doc/mysql/de/ANSI_diff_Triggers.html.

The manual used to say that a trigger is a stored procedure. Forget that, as you have seen, a trigger is just a trigger.

The manual used also to say that a trigger can delete from another table, or is activated when you delete a transaction, whatever that is supposed to mean. Forget that, the MySQL implementation can't do such things.

Lastly, the manual used to say that if you have triggers, then queries will be slower. Forget that too, triggers have no effect on queries.

Bugs

On December 14 2004, I did an “Advanced Search” in http://bugs.mysql.com for 'trigger' or 'triggers', I found that there were 17 active bugs as of that date. Of course they might disappear before you read this, but just in case they haven't, I'll mention the important ones. If they're still there, you'll have to work around them when you're trying triggers.

Bug#5859 DROP TABLE does not drop triggers.
When you drop a table, dropping the table's triggers should be automatic.

Bug#5892 Triggers have the wrong namespace.
You have to say “DROP TRIGGER <table name> . <trigger name>”.
The correct way is “DROP TRIGGER <trigger name>”.

Bug#5894 Triggers with altered tables cause corrupt databases.
Do not alter a table that has a trigger on it, until you know this is fixed.

Conclusion

You've come to the end of the book. I don't bother with a review or an index, since I'm sure you've had no trouble memorizing it all.

If you enjoyed this book, you should look for more in the “MySQL 5.0 New Features” series. The earlier book was “Stored Procedures”. The next will be “Views”.

Thank you very much for your attention. If you have any comments about this book, please send them to one of the MySQL forums at:
http://forums.mysql.com

About MySQL

MySQL AB develops and supports a family of high performance, affordable database servers and tools. The company's flagship product is MySQL, the world's most popular open source database, with more than six million active installations. Many of the world's largest organizations, including Yahoo!, Sabre Holdings, The Associated Press, Suzuki and NASA are realizing significant cost savings by using MySQL to power high-volume Web sites, business-critical enterprise applications and packaged software.

With headquarters in Sweden and the United States – and operations around the world – MySQL AB supports both open source values and corporate customers' needs in a profitable, sustainable business. For more information about MySQL, please visit www.mysql.com.