Triggers in MYSQL

A trigger is a set of actions that are run automatically when a specified change operation (SQL INSERT, UPDATE, or DELETE statement) is performed on a specified table. Triggers are useful for tasks such as enforcing business rules, validating input data, and keeping an audit trail.

Uses for triggers:

- Enforce business rules
- Validate input data
- Generate a unique value for a newly-inserted row in a different file.
- Write to other files for audit trail purposes
- Query from other files for cross-referencing purposes
- Access system functions
- Replicate data to different files to achieve data consistency

Benefits of using triggers in business:

- Faster application development. Because the database stores triggers, you do not have to code the trigger actions into each database application.
- Global enforcement of business rules. Define a trigger once and then reuse it for any application that uses the database.
- Easier maintenance. If a business policy changes, you need to change only the corresponding trigger program instead of each application program.
- Improve performance in client/server environment. All rules run on the server before the result returns.

How to create MySQL triggers?

A trigger is a named database object that is associated with a table, and it activates when a particular event (e.g. an insert, update or delete) occurs for the

table. The statement CREATE TRIGGER creates a new trigger in MySQL. Here is the syntax :

Syntax:

```
CREATE
[DEFINER = { user | CURRENT_USER }]
TRIGGER trigger_name
trigger_time trigger_event
ON tbl_name FOR EACH ROW
trigger_body
trigger_time: { BEFORE | AFTER }
trigger_event: { INSERT | UPDATE | DELETE }
```

Explanation:

DEFINER clause: The DEFINER clause specifies the MySQL account to be used when checking access privileges at trigger activation time. If a user value is given, it should be a MySQL account specified as 'user_name'@'host_name' (the same format used in the GRANT statement), CURRENT_USER, or CURRENT_USER().
The default DEFINER value is the user who executes the CREATE TRIGGER statement. This is the same as specifying DEFINER = CURRENT_USER explicitly.
If you specify the DEFINER clause, these rules determine the valid DEFINER user values:

- If you do not have the SUPER privilege, the only permitted user value is your own account, either specified literally or by using CURRENT_USER. You cannot set the definer to some other account.
- If you have the SUPER privilege, you can specify any syntactically valid account name. If the account does not actually exist, a warning is generated.
- Although it is possible to create a trigger with a nonexistent DEFINER account, it is not a good idea for such triggers to be activated until the account actually does exist. Otherwise, the behavior with respect to privilege checking is undefined.

trigger_name: All triggers must have unique names within a schema. Triggers in different schemas can have the same name.

trigger_time: trigger_time is the trigger action time. It can be BEFORE or AFTER to indicate that the trigger activates before or after each row to be modified.

trigger_event: trigger_event indicates the kind of operation that activates the trigger. These trigger_event values are permitted:

- The trigger activates whenever a new row is inserted into the table; for example, through INSERT, LOAD DATA, and REPLACE statements.
- The trigger activates whenever a row is modified; for example, through UPDATE statements.
- The trigger activates whenever a row is deleted from the table; for example, through DELETE and REPLACE statements. DROP TABLE and TRUNCATE TABLE statements on the table do not activate this trigger, because they do not use DELETE. Dropping a partition does not activate DELETE triggers, either.

tbl_name : The trigger becomes associated with the table named tbl_name, which must refer to a permanent table. You cannot associate a trigger with a TEMPORARY table or a view.

trigger_body: trigger_body is the statement to execute when the trigger activates. To execute multiple statements, use the BEGIN ... END compound statement construct. This also enables you to use the same statements that are permissible within stored routines.

Here is a simple example:

```
mysql> CREATE TRIGGER ins_sum BEFORE INSERT ON account
    -> FOR EACH ROW SET @sum = @sum + NEW.amount;
Query OK, 0 rows affected (0.06 sec)
```

In the above example, there is new keyword '**NEW**' which is a MySQL extension to triggers. There is two MySQL extension to triggers '**OLD**' and '**NEW**'. OLD and NEW are not case sensitive.

- Within the trigger body, 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.

- In a 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.
- In a DELETE trigger, only OLD.col_name can be used; there is no new row.

A column named with OLD is read only. You can refer to it (if you have the SELECT privilege), but not modify it. You can refer to a column named with NEW 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. This means you can use a trigger to modify the values to be inserted into a new row or used to update a row. (Such a SET statement has no effect in an AFTER trigger because the row change will have already occurred.)

MySQL Trigger : Example AFTER INSERT

In the following example, we have two tables: emp_details and log_emp_details. To insert some information into log_ emp_details table (which have three fields employee id and salary and edttime) every time, when an INSERT happen into emp_details table we have used the following trigger :

```
DELIMITER
$$
USE `hr`
$$
CREATE
DEFINER=`root`@`127.0.0.1`
TRIGGER `hr`.`emp_details_AINS`
AFTER INSERT ON `hr`.`emp_details`
FOR EACH ROW
-- Edit trigger body code below this line. Do not edit lines
above this one
BEGIN
INSERT INTO log_emp_details
VALUES(NEW.employee_id, NEW.salary, NOW());
END$$
```

Records of the table (on some columns) : **emp_details**

```
mysql> SELECT EMPLOYEE_ID, FIRST_NAME, LAST_NAME, JOB_ID,
SALARY, COMMISSION_PCT FROM emp_details;
+-------------+------------+-----------+---------+----------+---
-------------+
| EMPLOYEE_ID | FIRST_NAME | LAST_NAME | JOB_ID  | SALARY   |
COMMISSION_PCT |
```

```
+-------------+-----------+----------+---------+----------+---
-------------+
|         100 | Steven     | King      | AD_PRES | 24000.00 |
0.10 |
|         101 | Neena      | Kochhar   | AD_VP   | 17000.00 |
0.50 |
|         102 | Lex        | De Haan   | AD_VP   | 17000.00 |
0.50 |
|         103 | Alexander  | Hunold    | IT_PROG |  9000.00 |
0.25 |
|         104 | Bruce      | Ernst     | IT_PROG |  6000.00 |
0.25 |
|         105 | David      | Austin    | IT_PROG |  4800.00 |
0.25 |
+-------------+-----------+----------+---------+----------+---
-------------+
6 rows in set (0.00 sec)
```

Records of the table (all columns) : **log_emp_details**

```
mysql> SELECT * FROM log_emp_details;
+-------------+----------+---------------------+
| emp_details | SALARY   | EDTTIME             |
+-------------+----------+---------------------+
|         100 | 24000.00 | 2011-01-15 00:00:00 |
|         101 | 17000.00 | 2010-01-12 00:00:00 |
|         102 | 17000.00 | 2010-09-22 00:00:00 |
|         103 |  9000.00 | 2011-06-21 00:00:00 |
|         104 |  6000.00 | 2012-07-05 00:00:00 |
|         105 |  4800.00 | 2011-06-21 00:00:00 |
+-------------+----------+---------------------+
6 rows in set (0.02 sec)
```

Now insert one record in emp_details table see the records both in emp_details and log_emp_details tables :

```
mysql> INSERT INTO emp_details VALUES(236, 'RABI', 'CHANDRA',
'RABI','590.423.45700', '2013-01-12', 'AD_VP', 15000, .5);
Query OK, 1 row affected (0.07 sec)
mysql> SELECT EMPLOYEE_ID, FIRST_NAME, LAST_NAME, JOB_ID,
SALARY, COMMISSION_PCT FROM emp_details;
+-------------+-----------+----------+---------+----------+---
-------------+
| EMPLOYEE_ID | FIRST_NAME | LAST_NAME | JOB_ID  | SALARY   |
COMMISSION_PCT |
+-------------+-----------+----------+---------+----------+---
-------------+
```

```
|         100 | Steven     | King      | AD_PRES | 24000.00 |
0.10 |
|         101 | Neena      | Kochhar   | AD_VP   | 17000.00 |
0.50 |
|         102 | Lex        | De Haan   | AD_VP   | 17000.00 |
0.50 |
|         103 | Alexander  | Hunold    | IT_PROG |  9000.00 |
0.25 |
|         104 | Bruce      | Ernst     | IT_PROG |  6000.00 |
0.25 |
|         105 | David      | Austin    | IT_PROG |  4800.00 |
0.25 |
|         236 | RABI       | CHANDRA   | AD_VP   | 15000.00 |
0.50 |
+-------------+------------+-----------+---------+----------+---
-------------+
7 rows in set (0.00 sec)
mysql> SELECT * FROM log_emp_details;
+-------------+----------+---------------------+
| emp_details | SALARY   | EDTTIME             |
+-------------+----------+---------------------+
|         100 | 24000.00 | 2011-01-15 00:00:00 |
|         101 | 17000.00 | 2010-01-12 00:00:00 |
|         102 | 17000.00 | 2010-09-22 00:00:00 |
|         103 |  9000.00 | 2011-06-21 00:00:00 |
|         104 |  6000.00 | 2012-07-05 00:00:00 |
|         105 |  4800.00 | 2011-06-21 00:00:00 |
|         236 | 15000.00 | 2013-07-15 16:52:24 |
+-------------+----------+---------------------+
7 rows in set (0.00 sec)
```

MySQL Trigger : Example BEFORE INSERT

In the following example, before insert a new record in emp_details table, a trigger check the column value of FIRST_NAME, LAST_NAME, JOB_ID and
- If there are any space(s) before or after the FIRST_NAME, LAST_NAME, TRIM() function will remove those.
- The value of the JOB_ID will be converted to upper cases by UPPER() function.

Here is the trigger code :

```
USE `hr`;
DELIMITER
$$
CREATE TRIGGER `emp_details_BINS`
BEFORE INSERT
```

```
ON emp_details FOR EACH ROW
-- Edit trigger body code below this line. Do not edit lines
above this one
BEGIN
SET NEW.FIRST_NAME = TRIM(NEW.FIRST_NAME);
SET NEW.LAST_NAME = TRIM(NEW.LAST_NAME);
SET NEW.JOB_ID = UPPER(NEW.JOB_ID);END;
$$
```

Now insert a row into emp_details table (check the FIRST_NAME, LAST_NAME, JOB_ID columns) :

```
mysql> INSERT INTO emp_details VALUES (334, ' Ana ', ' King',
'ANA', '690.432.45701', '2013-02-05', 'it_prog', 17000, .50);
Query OK, 1 row affected (0.04 sec)
```

Now list the following fields of emp_details :

```
mysql> SELECT EMPLOYEE_ID, FIRST_NAME, LAST_NAME, JOB_ID,
SALARY, COMMISSION_PCT FROM emp_details;
+-------------+------------+-----------+---------+----------+---
-------------+
| EMPLOYEE_ID | FIRST_NAME | LAST_NAME | JOB_ID  | SALARY   |
COMMISSION_PCT |
+-------------+------------+-----------+---------+----------+---
-------------+
|         100 | Steven     | King      | AD_PRES | 24000.00 |
0.10 |
|         101 | Neena      | Kochhar   | AD_VP   | 17000.00 |
0.50 |
|         102 | Lex        | De Haan   | AD_VP   | 17000.00 |
0.50 |
|         103 | Alexander  | Hunold    | IT_PROG |  9000.00 |
0.25 |
|         104 | Bruce      | Ernst     | IT_PROG |  6000.00 |
0.25 |
|         105 | David      | Austin    | IT_PROG |  4800.00 |
0.25 |
|         236 | RABI       | CHANDRA   | AD_VP   | 15000.00 |
0.50 |
|         334 | Ana        | King      | IT_PROG | 17000.00 |
0.50 |
+-------------+------------+-----------+---------+----------+---
-------------+
8 rows in set (0.00 sec)
```

See the last row :

FIRST_NAME - > ' Ana ' has changed to 'Ana'
LAST_NAME - > ' King' has changed to 'King'
JOB_ID - > ' it_prog' has changed to 'IT_PROG'

MySQL Trigger : Example AFTER UPDATE

We have two tables student_mast and stu_log. student_mast have three columns STUDENT_ID, NAME, ST_CLASS. stu_log table has two columns user_id and description.

```
mysql> SELECT * FROM STUDENT_MAST;
+------------+------------------+----------+
| STUDENT_ID | NAME             | ST_CLASS |
+------------+------------------+----------+
|          1 | Steven King      |        7 |
|          2 | Neena  Kochhar   |        8 |
|          3 | Lex  De Haan     |        8 |
|          4 | Alexander Hunold |       10 |
+------------+------------------+----------+
4 rows in set (0.00 sec)
```

Let we promote all the students in next class i.e. 7 will be 8, 8 will be 9 and so on. After updating a single row in student_mast table a new row will be inserted in stu_log table where we will store the current user id and a small description regarding the current update. Here is the trigger code :

```
-- Full Trigger
DDL Statements
-- Note: Only CREATE TRIGGER statements are allowed
DELIMITER
$$
USE `test`
$$
CREATE
DEFINER=`root`@`127.0.0.1`
TRIGGER `test`.`student_mast_AUPD`
AFTER UPDATE
ON `test`.`student_mast`FOR EACH ROW
-- Edit trigger body code below this line. Do not edit lines
above this one
BEGIN
INSERT into stu_log VALUES (user(), CONCAT('Update Student
Record ',
         OLD.NAME,' Previous Class :',OLD.ST_CLASS,' Present
Class ',
```

```
         NEW.st_class));
END
$$
```

After update STUDENT_MAST table :

```
mysql> UPDATE STUDENT_MAST SET ST_CLASS = ST_CLASS + 1;
Query OK, 4 rows affected (0.20 sec)
Rows matched: 4
Changed: 4
Warnings: 0
```

The trigger show you the updated records in 'stu_log'. Here is the latest position of STUDENT_MAST and STU_LOG tables :

```
mysql> SELECT * FROM STUDENT_MAST;
+------------+------------------+----------+
| STUDENT_ID | NAME             | ST_CLASS |
+------------+------------------+----------+
|          1 | Steven King      |        8 |
|          2 | Neena  Kochhar   |        9 |
|          3 | Lex  De Haan     |        9 |
|          4 | Alexander Hunold |       11 |
+------------+------------------+----------+
4 rows in set (0.00 sec)mysql> SELECT * FROM STU_LOG;
+---------------+-----------------------------------------------
-----------------------------+
| user_id        | description
|
+---------------+-----------------------------------------------
-----------------------------+
| root@localhost | Update Student Record Steven King Previous
Class :7 Present Class 8       |
| root@localhost | Update Student Record Neena  Kochhar Previous
Class :8 Present Class 9    |
| root@localhost | Update Student Record Lex  De Haan Previous
Class :8 Present Class 9      |
| root@localhost | Update Student Record Alexander Hunold
Previous Class :10 Present Class 11|
+---------------+-----------------------------------------------
-----------------------------+
4 rows in set (0.00 sec)
```

MySQL Trigger : Example BEFORE UPDATE

We have a table student_marks with 10 columns and 4 rows. There are data only in STUDENT_ID and NAME columns.

```
mysql> SELECT * FROM STUDENT_MARKS;
```

```
+------------+------------------+------+------+------+------+---
---+-------+-----------+-------+
| STUDENT_ID | NAME             | SUB1 | SUB2 | SUB3 | SUB4 |
SUB5 | TOTAL | PER_MARKS | GRADE |
+------------+------------------+------+------+------+------+---
---+-------+-----------+-------+
|          1 | Steven King      |    0 |    0 |    0 |    0 |
0 |     0 |      0.00 |       |
|          2 | Neena  Kochhar   |    0 |    0 |    0 |    0 |
0 |     0 |      0.00 |       |
|          3 | Lex  De Haan     |    0 |    0 |    0 |    0 |
0 |     0 |      0.00 |       |
|          4 | Alexander Hunold |    0 |    0 |    0 |    0 |
0 |     0 |      0.00 |       |
+------------+------------------+------+------+------+------+---
---+-------+-----------+-------+
4 rows in set (0.00 sec)
```

Now the exam is over and we have received all subject marks, now we will update the table, total marks of all subject, the percentage of total marks and grade will be automatically calculated. For this sample calculation, the following conditions are assumed :

Total Marks (will be stored in TOTAL column) : TOTAL = SUB1 + SUB2 + SUB3 + SUB4 + SUB5

Percentage of Marks (will be stored in PER_MARKS column) : PER_MARKS = (TOTAL)/5

Grade (will be stored GRADE column) :
- If PER_MARKS>=90 -> 'EXCELLENT'
- If PER_MARKS>=75 AND PER_MARKS<90 -> 'VERY GOOD'
- If PER_MARKS>=60 AND PER_MARKS<75 -> 'GOOD'
- If PER_MARKS>=40 AND PER_MARKS<60 -> 'AVERAGE'
- If PER_MARKS<40-> 'NOT PROMOTED'

Here is the code :

```
mysql> UPDATE STUDENT_MARKS SET SUB1 = 54, SUB2 = 69, SUB3 = 89,
SUB4 = 87, SUB5 = 59 WHERE STUDENT_ID = 1;
Query OK, 1 row affected (0.05 sec)
Rows matched: 1
Changed: 1
```

```
Warnings: 0
```

Let update the marks of a student :

```
USE `test`;
DELIMITER
$$
CREATE TRIGGER `student_marks_BUPD`
BEFORE UPDATE
ON student_marks FOR EACH ROW
-- Edit trigger body code below this line. Do not edit lines
above this one
BEGIN
SET NEW.TOTAL = NEW.SUB1 + NEW.SUB2 + NEW.SUB3 + NEW.SUB4 +
NEW.SUB5;
SET NEW.PER_MARKS = NEW.TOTAL/5;
IF NEW.PER_MARKS >=90 THEN
SET NEW.GRADE = 'EXCELLENT';
ELSEIF NEW.PER_MARKS>=75 AND NEW.PER_MARKS<90 THEN
SET NEW.GRADE = 'VERY GOOD';
ELSEIF NEW.PER_MARKS>=60 AND NEW.PER_MARKS<75 THEN
SET NEW.GRADE = 'GOOD';
ELSEIF NEW.PER_MARKS>=40 AND NEW.PER_MARKS<60 THEN
SET NEW.GRADE = 'AVERAGE';
ELSESET NEW.GRADE = 'NOT PROMOTED';
END IF;
END;
$$
```

Now check the STUDENT_MARKS table with updated data. The trigger show you the updated records in 'stu_log'.

```
mysql> SELECT * FROM STUDENT_MARKS;
+------------+------------------+------+------+------+------+---
---+-------+----------+-------+
| STUDENT_ID | NAME             | SUB1 | SUB2 | SUB3 | SUB4 |
SUB5 | TOTAL | PER_MARKS | GRADE |
+------------+------------------+------+------+------+------+---
---+-------+----------+-------+
|          1 | Steven King      |   54 |   69 |   89 |   87 |
59 |   358 |     71.60 | GOOD  |
|          2 | Neena  Kochhar   |    0 |    0 |    0 |    0 |
0 |     0 |      0.00 |       |
|          3 | Lex  De Haan     |    0 |    0 |    0 |    0 |
0 |     0 |      0.00 |       |
|          4 | Alexander Hunold |    0 |    0 |    0 |    0 |
0 |     0 |      0.00 |       |
```

```
+-----------+-----------------+------+------+------+------+---
---+-------+----------+-------+
4 rows in set (0.00 sec)
```

MySQL Trigger : Example AFTER DELETE

In our 'AFTER UPDATE' example, we had two tables student_mast and stu_log. student_mast have three columns STUDENT_ID, NAME, ST_CLASS and stu_log table has two columns user_id and description. We want to store some information in stu_log table after a delete operation happened on student_mast table. Here is the trigger :

```
USE `test`;
DELIMITER
$$
CREATE TRIGGER `student_mast_ADEL`
AFTER DELETE ON student_mast FOR EACH ROW
-- Edit trigger body code below this line. Do not edit lines
above this one
BEGIN
INSERT into stu_log VALUES (user(), CONCAT('Update Student
Record ',
         OLD.NAME,' Clas :',OLD.ST_CLASS, '-> Deleted on ',
NOW()));
END;
$$
```

Let delete a student from STUDENT_MAST.

```
mysql> DELETE FROM STUDENT_MAST WHERE STUDENT_ID = 1;
Query OK, 1 row affected (0.06 sec)
```

Here is the latest position of STUDENT_MAST, STU_LOG tables :

```
mysql> SELECT * FROM STUDENT_MAST;
+------------+-----------------+----------+
| STUDENT_ID | NAME            | ST_CLASS |
+------------+-----------------+----------+
|          2 | Neena  Kochhar  |        9 |
|          3 | Lex  De Haan    |        9 |
|          4 | Alexander Hunold |       11 |
+------------+-----------------+----------+
3 rows in set (0.00 sec)
mysql> SELECT * FROM STU_LOG;
+---------------+-----------------------------------------------
------------------------------+
| user_id        | description
|
```

```
+---------------+-----------------------------------------
------------------------------+
| root@localhost | Update Student RecordSteven King Previous
Class :7 Present Class 8          |
| root@localhost | Update Student RecordNeena  Kochhar Previous
Class :8 Present Class 9       |
| root@localhost | Update Student RecordLex  De Haan Previous
Class :8 Present Class 9         |
| root@localhost | Update Student RecordAlexander Hunold
Previous Class :10 Present Class 11   |
| root@localhost | Update Student Record Steven King Clas :8->
Deleted on 2013-07-16 15:35:30  |
+---------------+-----------------------------------------
------------------------------+
5 rows in set (0.00 sec)
```

How MySQL handle errors during trigger execution?

- 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 any BEFORE triggers and the row operation execute successfully.
- An error during either a BEFORE or AFTER trigger results in failure of the entire statement that caused trigger invocation.
- For transactional tables, failure of a statement should cause a rollback of all changes performed by the statement.

Delete a MySQL trigger

To delete or destroy a trigger, use a DROP TRIGGER statement. You must specify the schema name if the trigger is not in the default (current) schema :

```
DROP TRIGGER [IF EXISTS] [schema_name.]trigger_nam
```

if you drop a table, any triggers for the table are also dropped.