Triggers

A SQL trigger is an SQL statement, or a set of SQL statements, which is stored to be activated or fired when an event associated with a database table occurs. The event can be any event including INSERT, UPDATE, or DELETE.

Sometimes a trigger is referred to as a special kind of stored procedure in terms of procedural code inside its body. The difference between a trigger and a stored procedure is that a trigger is activated or called when an event occurs in a database table, a stored procedure must be called explicitly. For example, you can have some business logic to do before or after inserting a new record in a database table.

Advantages:

- Provides an alternative way to check integrity.
- Can catch errors in business logic.
- Business rules enforced with changes made to the database.
- Provides an alternative way to run scheduled tasks. Don't have to wait to run scheduled tasks. Handle tasks before or after changes made to database tables.
- Useful when auditing data changes in database tables.

Disadvantages:

- Can provide extended validation, but cannot replace all validation rules. Some simple validations can be done at the application level. For example, you can validate input checking at the client side by using JavaScript, or at the server side by server-side script using PHP or ASP.NET.
- Executes invisibly from client-application which connects to the database server so it may be difficult to figure out what happens to the underlying database layer.
- Runs with changes made to their associated tables; therefore, it adds an additional workload to the server and can cause systems to run more slowly.

Triggers or stored procedures? It depends on the situation.

Creating Triggers (require the following five items**):**

1. Name: Unique
2. Invoked: AFTER or BEFORE event
3. Action: which it responds to (e.g., INSERT, UPDATE, or DELETE)
4. Table: which it is to be associated
5. Body: SQL statements

Syntax:

```
CREATE TRIGGER trigger_name          # trigger name
{BEFORE | AFTER}                     # when trigger activates
{INSERT | UPDATE | DELETE}           # what statement activates trigger
ON tbl_name                          # associated table
FOR EACH ROW trigger_body            # what trigger does
```

Example:
Every time a user adds/updates/deletes an inventory record, insertion data will be logged to a separate table (log/audit), including the user's MySQL username, suitable notes (e.g., "record added"), and a modification time stamp.
Note: this demo also includes data validation for validating proper users.

```
-- Demo: drop database in order to avoid fk conflict with inventory
drop database if exists triggers;
create database if not exists triggers;
use triggers;

-- Create inventory table:
drop table if exists inventory;
create table if not exists inventory
(
  ivt_id int not null auto_increment,
  ivt_desc varchar(30),
  ivt_notes varchar(100),
  primary key (ivt_id)
);

-- Create log/audit table:
drop table if exists inventory_log;
create table if not exists inventory_log
(
  ivl_id int not null auto_increment,
  ivt_id int null COMMENT 'Allow null fk to demo parent table deletions
without cascading.',
  ivl_user varchar(30),
  ivl_date timestamp,
  ivl_notes varchar(100),
  primary key (ivl_id),

  constraint fk_ivt_id
    foreign key (ivt_id)
    references inventory (ivt_id)
    on delete set null
    on update cascade
);

-- test tables before INSERT trigger
select * from inventory;
select * from inventory_log;

-- %%%%% Create INSERT trigger using NEW keyword: %%%%%
drop trigger if exists trg_inventory_after_insert;

-- temporarily redefine delimiter
delimiter //
create trigger trg_inventory_after_insert
  AFTER INSERT on inventory
  FOR EACH ROW
  BEGIN
```

```
  INSERT into inventory_log
  (ivt_id, ivl_user, ivl_date, ivl_notes)
  values (NEW.ivt_id, user(), now(), concat("Inventory item ", NEW.ivt_id, "
after insert."));

      /*
      -- Or, using data validation: here, preventing log entry from user w/o
proper permissions
      IF STRCMP(user(), 'youruserid@localhost') != 0
      THEN
      -- Note: Not all MySQL error numbers have corresponding SQLSTATE
values. In these cases, 'HY000' (general error) is used.
         set @sql_error = concat('Improper permissions for ', user());
            SIGNAL SQLSTATE 'HY000'
              SET MESSAGE_TEXT = @sql_error;
      ELSE
            INSERT into inventory_log
            (ivt_id, ivl_user, ivl_date, ivl_notes)
      values
            (NEW.ivt_id, user(), now(), concat("Inventory item ", NEW.ivt_id,
" after insert."));
      END IF;
  */

  END //
delimiter ;

-- test trigger (add new inventory record):
insert into inventory(ivt_id, ivt_desc, ivt_notes)
values (null, "vacuum", "first item stocked");

-- test tables after INSERT trigger
select * from inventory;
select * from inventory_log;

-- drop trigger (only if no longer using):
-- drop trigger if exists trg_inventory_after_insert;

-- %%%%% Create BEFORE UPDATE trigger using OLD keyword: %%%%%
drop trigger if exists trg_inventory_before_update;

-- temporarily redefine delimiter
delimiter //
create trigger trg_inventory_before_update
  BEFORE UPDATE on inventory
  FOR EACH ROW
  BEGIN
  INSERT into inventory_log
  (ivt_id, ivl_user, ivl_date, ivl_notes)
  values (OLD.ivt_id, user(), now(), concat("Inventory item ", OLD.ivt_id, "
description: ", OLD.ivt_desc, ", before update."));
  END //
delimiter ;
```

```
-- test tables before UPDATE trigger
select * from inventory;
select * from inventory_log;

-- test trigger (update inventory record):
-- update inventory set ivt_desc='old vacuum' where ivt_id=1;

-- test tables after UPDATE trigger
select * from inventory;
select * from inventory_log;

-- drop trigger (only if no longer using):
-- drop trigger if exists trg_inventory_before_update;

-- %%%%% Create AFTER UPDATE trigger using NEW keyword: %%%%%
drop trigger if exists trg_inventory_after_update;

-- temporarily redefine delimiter
delimiter //
create trigger trg_inventory_after_update
  AFTER UPDATE on inventory
  FOR EACH ROW
  BEGIN
  INSERT into inventory_log
  (ivt_id, ivl_user, ivl_date, ivl_notes)
  values (NEW.ivt_id, user(), now(), concat("Inventory item ", NEW.ivt_id, "
description: ", NEW.ivt_desc, ", after update."));
  END //
delimiter ;

-- test tables after UPDATE trigger
select * from inventory;
select * from inventory_log;

-- test trigger (update inventory record):
update inventory set ivt_desc='new and improved vacuum' where ivt_id=1;

-- test tables after UPDATE trigger
select * from inventory;
select * from inventory_log;

-- drop trigger (only if no longer using):
-- drop trigger if exists trg_inventory_after_update;
```

```
-- %%%%% Create DELETE trigger using OLD keyword: %%%%%
drop trigger if exists trg_inventory_after_delete;

-- temporarily redefine delimiter
delimiter //
create trigger trg_inventory_after_delete
  AFTER DELETE on inventory
  FOR EACH ROW
  BEGIN
  INSERT into inventory_log
  (ivt_id, ivl_user, ivl_date, ivl_notes)
  values (NULL, user(), now(), concat("Inventory item ", OLD.ivt_id, " after
delete."));
  END //
delimiter ;

-- test tables before DELETE trigger
select * from inventory;
select * from inventory_log;

-- test trigger (delete inventory record):
delete from inventory where ivt_id=1;

-- test tables after DELETE trigger
select * from inventory;
select * from inventory_log;

-- drop trigger (only if no longer using):
-- drop trigger if exists trg_inventory_after_delete;
```

Triggers are created using the CREATE TRIGGER statement.
http://dev.mysql.com/doc/refman/5.6/en/trigger-syntax.html
CREATE TRIGGER is used to create the new trigger named **trg_inventory_mod**. Triggers can be executed before or after an operation occurs, and here AFTER INSERT is specified so the trigger will execute after a successful INSERT statement has been executed. The trigger then specifies FOR EACH ROW and the code to be executed for each inserted row. In this example, user modification info will be inserted into the log table, for each row inserted into the inventory table.
To test this trigger, use the INSERT statement to add one or more rows to inventory; then check and select all the records in the log table.

Note:
Triggers are only supported on tables, **not** on views (or temporary tables).

Triggers are defined per time, per event, per table, and only one trigger per time, per event, per table is allowed. As such, up to six triggers are supported per table (before and after each of INSERT, UPDATE, and DELETE). A single trigger cannot be associated with multiple events or multiple tables, so if you need a trigger to be executed for both INSERT and UPDATE operations, you'll need to define two triggers.

Similarly, a trigger can only be associated with a table and defined to fire when an INSERT, DELETE or UPDATE statement is performed on the table. MySQL does not permit two triggers with the same trigger timing (BEFORE or AFTER) and trigger event or statement (INSERT, DELETE, or UPDATE) to be defined on a table. For example, you cannot define two BEFORE INSERT or two AFTER UPDATE triggers for a table. All triggers defined on MySQL

are row triggers, which means that the action defined for the triggers is executed for each row affected by the triggering statement.

OLD and NEW are MySQL extensions to triggers.
The OLD and NEW keywords enable you to access columns in the rows affected by a trigger. (OLD and NEW are not case sensitive.)

- 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 (if you have the SELECT privilege), 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. This means you can use a trigger to modify the values to be inserted into a new row or that are used to update a row.

In a BEFORE trigger, the NEW value for an AUTO_INCREMENT column is 0, not the automatically generated sequence number that will be generated when the new record actually is inserted.

Display triggers:

```
mysql> show triggers;
```

References

https://dev.mysql.com/doc/refman/5.5/en/trigger-syntax.html
https://dev.mysql.com/doc/refman/5.7/en/trigger-syntax.html
http://www.mysqltutorial.org/create-the-first-trigger-in-mysql.aspx
http://www.w3resource.com/mysql/mysql-triggers.php