Write and execute suitable database triggers .Consider row level and statement level triggers.

In MySQL, a trigger is a set of SQL statements that is invoked automatically when a change is made to the data on the associated table. A trigger can be defined to be invoked either before or after the data is changed by INSER, UPDATE or DELETE statement. Before MySQL version 5.7.2, it was possible to define maximum six triggers for each table.

BEFORE INSERT - activated before data is inserted into the table.
AFTER INSERT - activated after data is inserted into the table.
BEFORE UPDATE - activated before data in the table is updated.
AFTER UPDATE - activated after data in the table is updated.
BEFORE DELETE - activated before data is removed from the table.
AFTER DELETE - activated after data is removed from the table.

However, from MySQL version 5.7.2+, you can define multiple triggers for the same trigger event and action time.

When you use a statement that does not use INSERT, DELETE or UPDATE statement to change data in a table, the triggers associated with the table are not invoked. For example, the TRUNCATE statement removes all data of a table but does not invoke the trigger associated with that table.

There are some statements that use the `INSERT` statement behind the scenes such as REPLACE statement or LOAD DATA statement. If you use these statements, the corresponding triggers associated with the table are invoked.

You must use a unique name for each trigger associated with a table. However, you can have the same trigger name defined for different tables though it is a good practice.

MySQL Trigger Synatx
Syntax :

```
CREATE TRIGGER trigger_name trigger_time trigger_event
ON table_name
FOR EACH ROW
BEGIN
.....
END;
```

You put the trigger name after the `CREATE TRIGGER` statement. The trigger name should follow the naming convention `[trigger time]_[table name]_[trigger event]`, for example `before_employees_update`.

Trigger activation time can be `BEFORE` or `AFTER`. You must specify the activation time when you define a trigger. You use the `BEFORE` keyword if you want to process action prior to the change is made on the table and `AFTER` if you need to process action after the change is made.

The trigger event can be `INSERT`, `UPDATE` or `DELETE`. This event causes the trigger to be invoked. A trigger only can be invoked by one event. To define a trigger that is invoked by multiple

events, you have to define multiple triggers, one for each event.

A trigger must be associated with a specific table. Without a table trigger would not exist therefore you have to specify the table name after the `ON` *keyword.*

You place the SQL statements between `BEGIN` *and* `END` *block. This is where you define the logic for the trigger.*

MySQL Trigger Example on emp table

First, create a new table named `employees_audit` to keep the changes of the `employee` table.

CREATE TABLE employees_audit (
id INT AUTO_INCREMENT PRIMARY KEY,
employeeNumber INT NOT NULL,
firstname VARCHAR(50) NOT NULL,
changedat DATETIME DEFAULT NULL,
action VARCHAR(50) DEFAULT NULL
);

Next, create a `BEFORE UPDATE` *trigger that is invoked before a change is made to the* `employees` *table.*

```
DELIMITER $$
CREATE TRIGGER before_employee_update
BEFORE UPDATE ON emp
FOR EACH ROW
BEGIN
INSERT INTO employees_audit
SET action = 'update',
employeeNumber = OLD.empno,
firstname = OLD.ename,
changedat = NOW();
END$$
DELIMITER ;
```

Inside the body of the trigger, we used the `OLD` keyword to access `empno` and `ename` column of the row affected by the trigger.

Notice that in a trigger defined for INSERT, you can use `NEW` keyword only. You cannot use the `OLD` keyword. However, in the trigger defined for DELETE, there is no new row so you can use the `OLD` keyword only. In the UPDATE trigger, `OLD` refers to the row before it is updated and `NEW` refers to the row after it is updated.

Then, to view all triggers in the current database, you use `SHOW TRIGGERS` *statement as follows:*

```
SHOW TRIGGERS;
```

After that, update the `employees` table to check whether the trigger is invoked.

```
UPDATE emp
```

```
SET
ename = 'SMITH'
WHERE
sal = 800;
```

Finally, to check if the trigger was invoked by the `UPDATE` statement, you can query the `employees_audit` table using the following query:

```
select * from employees_audit;
```

MySQL trigger limitations

MySQL triggers cover all features defined in the standard SQL. However, there are some limitations that you should know before using them in your applications.

MySQL triggers cannot:

1. Use SHOW, LOAD DATA, LOAD TABLE, BACKUP DATABASE, RESTORE, FLUSH and RETURN statements.
2. Use statements that commit or rollback implicitly or explicitly such as COMMIT, ROLLBACK , START TRANSACTION , LOCK/UNLOCK TABLES , ALTER , CREATE , DROP , RENAME.
3. Use prepared statements such as PREPAREand EXECUTE.
4. Use dynamic SQL statements.

From MySQL version 5.1.4, a trigger can call a stored procedure or stored function, which was a limitation is the previous versions.