Summary – Day 15

Triggers:

- It is a special type of stored procedure that is invoked automatically in response to an event. Each trigger is associated with a table, which is activated on any DML statement such as INSERT, UPDATE, or DELETE.

- A trigger is called a special procedure because it cannot be called directly like a stored procedure. The main difference between the trigger and procedure is that a trigger is called automatically when a data modification event is made against a table. In contrast, a stored procedure must be called explicitly.

- In MySQL, a trigger is a stored program invoked automatically in response to an event such as insert, update, or delete that occurs in the associated table. For example, you can define a trigger that is invoked automatically before a new row is inserted into a table.

- MySQL supports triggers that are invoked in response to the INSERT, UPDATE or DELETE event.

- The SQL standard defines two types of triggers: row-level triggers and statement-level triggers.

- A row-level trigger is activated for each row that is inserted, updated, or deleted. For example, if a table has 100 rows inserted, updated, or deleted, the trigger is automatically invoked 100 times for the 100 rows affected.

- A statement-level trigger is executed once for each transaction regardless of how many rows are inserted, updated, or deleted.

- MySQL supports only row-level triggers. It doesn't support statement-level triggers.

Types:

1. Before Insert: It is activated before the insertion of data into the table.
2. After Insert: It is activated after the insertion of data into the table.
3. Before Update: It is activated before the update of data in the table.
4. After Update: It is activated after the update of the data in the table.
5. Before Delete: It is activated before the data is removed from the table.
6. After Delete: It is activated after the deletion of data from the table.

Syntax:

```
DELIMITER //
CREATE TRIGGER trigger_name
   (AFTER | BEFORE) (INSERT | UPDATE | DELETE)
      ON table_name FOR EACH ROW
      BEGIN
     --variable declarations
     --trigger code
     END;
```

Parameters Explanation:

The create trigger syntax contains the following parameters:

- trigger_name: It is the name of the trigger that we want to create. It must be written after the CREATE TRIGGER statement. It is to make sure that the trigger name should be unique within the schema.

- trigger_time: It is the trigger action time, which should be either BEFORE or AFTER. It is the required parameter while defining a trigger. It indicates that the trigger will be invoked before or after each row modification occurs on the table.

- trigger_event: It is the type of operation name that activates the trigger. It can be either INSERT, UPDATE, or DELETE operation. The trigger can invoke only one event at one time. If we want to define a trigger which is invoked by multiple events, it is required to define multiple triggers, and one for each event.

- table_name: It is the name of the table to which the trigger is associated. It must be written after the ON keyword. If we did not specify the table name, a trigger would not exist.

- BEGIN END Block: Finally, we will specify the statement for execution when the trigger is activated. If we want to execute multiple statements, we will use the BEGIN END block that contains a set of queries to define the logic for the trigger.

- The trigger body can access the column's values, which are affected by the DML statement. The NEW and OLD modifiers are used to distinguish the column values BEFORE and AFTER the execution of the DML statement. We can use the column name with NEW and OLD modifiers as OLD.col_name and NEW.col_name. The OLD.column_name indicates the column of an existing row before the updation or deletion occurs. NEW.col_name indicates the column of a new row that will be inserted or an existing row after it is updated.

- For example, suppose we want to update the column name message_info using the trigger. In the trigger body, we can access the column value before the update as OLD.message_info and the new value NEW.message_info.

SHOW TRIGGERS:

The show or list trigger is much needed when we have many databases that contain various tables. Sometimes we have the same trigger names in many databases; this query plays an important role in that case. We can get the trigger information in the database server using the below statement. This statement returns all triggers in all databases.

```
SHOW TRIGGERS;
```

DROP TRIGGER:

We can drop/delete/remove a trigger in MySQL using the DROP TRIGGER statement. You must be very careful while removing a trigger from the table. Because once we have deleted the trigger, it cannot be recovered. If a trigger is not found, the DROP TRIGGER statement throws an error.

```
DROP TRIGGER [IF EXISTS] [schema_name.]trigger_name;
```

Why we need/use triggers in MySQL ?

- Triggers help us to enforce business rules.
- Triggers help us to validate data even before they are inserted or updated.
- Triggers help us to keep a log of records like maintaining audit trails in tables.
- SQL triggers provide an alternative way to check the integrity of data.
- Triggers provide an alternative way to run the scheduled task.
- Triggers increases the performance of SQL queries because it does not need to compile each time the query is executed.
- Triggers reduce the client-side code that saves time and effort.
- Triggers help us to scale our application across different platforms.
- Triggers are easy to maintain.

Limitations:

- MySQL triggers do not allow to use of all validations; they only provide extended validations. For example, we can use the NOT NULL, UNIQUE, CHECK and FOREIGN KEY constraints for simple validations.
- Triggers are invoked and executed invisibly from the client application. Therefore, it isn't easy to troubleshoot what happens in the database layer.
- Triggers may increase the overhead of the database server.