Lesson 9-Views and Triggers

Lesson 9 Concepts

1. Views are a saved query that returns a results table that can be interacted with
2. Triggers are a script that initiates BEFORE or AFTER a DML event (INSERT, UPDATE, DELETE)

Views are "virtual tables" – a saved query that returns a results table that can be interacted with.

Views are results tables that represent a snapshot of your data that matches specific query parameters. You can represent as data from a single table that the result of a complex join. The commands that support views are CREATE VIEW, ALTER VIEW and DROP VIEW.

```
CREATE VIEW name_of_view
AS SELECT column1
FROM table_name
WHERE condition;
```

Note that a view is not a database table. A view is a shortcut to run a query and get back a results table. Every time you interact with the view, you are running the saved query and getting updated results. Views are not stored data – just a stored query.

Advantages of VIEWS:

- Simplifies complex queries
- Reusability: Save frequently used queries as views for easy reference
- Protect data: An extra layer of security to limit or restrict data access

Limitations of VIEWS:

- UPDATE, INSERT and DELETE can be used with some views but not recommended
- No DISTINCT
- No GROUP BY or HAVING
- No aggregate functions

Altering a view:
ALTER VIEW name_of_view AS SELECT column1 FROM table_name WHERE condition;

Deleting a view:
DROP VIEW name_of_view;

Triggers are a script that runs when a DML (Data Manipulation Language) event occurs.
Triggers are automatically executed in response to a specific event on a specified table. These events can include data manipulation such as INSERT, UPDATE and DELETE and changes in table structure.

Trigger Events:

Trigger event	OLD	NEW
INSERT	No	yes
UPDATE	Yes	Yes
DELETE	Yes	no

Advantages of Triggers:

- Database integrity : Triggers provide an extra layer of checks for data integrity. For example, any invalid data will be prevented from being inserted or updated into the database.
- Auditing data changes. Triggers are useful for logging any changes in table data.
- Scheduled tasks: Triggers can automatically run scheduled tasks
-

Disadvantages of Triggers:

- Triggers do not provide any feedback to the database user so you will not be notified that there is an issue. Triggers can do things quietly.

- Difficult to troubleshoot because triggers run automatically

Basic syntax for creating a trigger:

CREATE TRIGGER trigger_name_event
BEFORE/AFTER INSERT/UPDATE/DELETE/
ON table_name
FOR EACH ROW
trigger body;

Start with **CREATE TRIGGER** statement and name the trigger. A way to name a trigger is description + event such as INSERT/UPDATE/DELETE so it is clear what the trigger does.

BEFORE/AFTER refers to when the trigger will occur – whether it is before or after an event **INSERT**, **UPDATE** or **DELETE.**

FOR EACH ROW means that each row is being updated when the trigger runs.

Trigger body refers to the SQL statements to be executed when the trigger is activated.

Types of triggers :

BEFORE INSERT
AFTER INSERT
BEFORE UPDATE
AFTER INSERT
BEFORE DELETE
AFTER DELETE

TRIGGERS – multiple statements

DELIMITER //
CREATE TRIGGER trigger_name_event
BEFORE/AFTER INSERT/UPDATE/DELETE/
ON table_name

FOR EACH ROW
BEGIN
-- trigger body: define the actions
END; //
DELIMITER ;

For triggers with multiple statements, the syntax requires a BEGIN and END as well as a DELIMITER.

DELIMITER:
A DELIMITER is required when there are multiple statements in the body. The role of the DELIMITER is to indicate where one statement ends and where the next one begins. For MySQL, the semi-colon is the default delimiter. However, when there are multiple statements, we need a way to group the statements together so that the SQL runs all the statements. This is where the DELIMITER steps in. By setting a custom DELIMITER such as // or $$ (in place of the semi-colon ;), we can define the block of code. Inside that code body, you can use semi-colons freely. After

BEGIN/END
BEGIN….END defines a trigger that will execute multiple statements and will require the use of a DELIMITER.

Trigger Order:

-before update > after update
-before insert > after insert

There is an order for triggers. Once you have set up triggers, it can become difficult to keep track the trigger and to troubleshoot if they are not in the proper order.
If there is no before update and you create an after update that is using the same tables and information, it will not work and will result in mull value.

If there is a null value, then you need to reset it to "0" and delete the triggers and try again in the order of the workflow.

References: https://dev.mysql.com/doc/refman/8.0/en/trigger-syntax.html