

# Trigger Definition, Use Cases and Explanation

**Trigger:** A named block of code that is executed or fired automatically when a certain type of SQL statement is executed in a particular database.

In SQLite, Triggers are driven(fired) by EVENT statements such as INSERT, UPDATE, or DELETE. And their order is defined by an execution constraint such as Trigger BEFORE an event or AFTER an EVENT.

## TRIGGER : 1

Here we are using our Trigger set 1 to create an Audit Trail Log.

- Our main purpose here is to keep a track record of any topic additions, switches or removals from any given courses.
- By keeping an Audit of what was deleted and when it was deleted. We could efficiently trace accidental edits, deletes OR provide proper information about the changes that were made to the database, in case it is needed.
- Every time a topic is deleted from a course, we will update the AuditCourseTopic Table which contains all essential information, needed to track the changes such as ids of Course and Topic, Name of Course and Topic, Timestamp of events, All these columns are acquired from a join of three tables: CourseTopic, Course and Topic.

For Example: If a new topic say “Duffing’s Oscillator” was introduced in our course “Mathematical Modelling”, and the trigger can be used to notify users of this new addition.

## Code and Operation:

1. The trigger `after_add_topic_to_course` fires after every addition to the linking table CourseTopic which links a course with a topic.
2. Every time a new value is entered. The trigger is fired and ‘NEW’ references can be exploited to point towards the new values, according to the following table. A NEW value is valid when AFTER statement is used with INSERT, UPDATE, and DELETE Statements

*INSERT*    NEW references are valid

*UPDATE*    NEW and OLD references are valid

*DELETE*    OLD references are valid

3. The specifics of the NEW Course/Topic are fetched from their respective tables via a Select statement, By using “AS” keyword we create an alias to the input value, this alias is then used to populate our Audit Table via INSERT statement.

```

CREATE TRIGGER if not EXISTS after_add_topic_to_course
AFTER
    INSERT ON CourseTopic FOR EACH ROW
BEGIN
    INSERT INTO AuditCourseTopic (
        course_id, title, topic_id, name, status,
        Timestamp
    )
select
    NEW.course_id as course_id,
    c.title as title,
    NEW.topic_id as topic_id,
    t.name as name,
    "New topic added to course" as status,
    CURRENT_TIMESTAMP as Timestamp
from
    CourseTopic ct,
    Course c,
    topic t
where
    ct.course_id = NEW.course_id
    AND ct.course_id = c.number
    AND ct.topic_id = t.tid;
END;

```

## Screenshots of execution and output from the set of triggers for INSERT UPDATE, and DELETE.

The screenshot shows a SQL IDE with a script editor and a results pane. The script editor contains the following SQL code:

```

93  /*INSERT*/
94  INSERT INTO CourseTopic VALUES('CS1014',101);
95
96
97  /*DELETE*/
98  DELETE FROM CourseTopic WHERE topic_id=97;
99
100
101  /*UPDATE*/
102  UPDATE CourseTopic SET topic_id=55 WHERE course_id='CS1015';
103
104  |
105  SELECT * From AuditCourseTopic;
106
107  /* The Purpose of this very basic trigger is to make every course number uppercase */
108
109
110

```

The results pane shows the output of the SQL execution. It includes a search bar with the text "new" and a table with 7 columns: auditID, course\_id, title, topic\_id, name, status, and Timestamp. The table contains 4 rows of data.

| auditID | course_id | title  | topic_id                | name | status                    | Timestamp                       |                     |
|---------|-----------|--------|-------------------------|------|---------------------------|---------------------------------|---------------------|
| 1       | 1         | CS1011 | Intergalactic SpaceWars | 101  | Blackholes and Hurricanes | New topic added to course       | 2020-07-21 20:52:45 |
| 2       | 2         | CS1011 | Intergalactic SpaceWars | 101  | Blackholes and Hurricanes | New topic added to course       | 2020-07-21 20:53:34 |
| 3       | 3         | CS1000 | idme                    | 101  | Blackholes and Hurricanes | Topic was changed in the course | 2020-07-21 20:53:58 |
| 4       | 4         | CS7200 | volutpat. Nulla         | 99   | Jamal Montgomery          | Topic was deleted from Course   | 2020-07-21 20:54:16 |

Execution finished without errors.  
Result: 4 rows returned in 13ms  
At line 104:  
SELECT \* From AuditCourseTopic;

## Trigger 2: A very basic trigger to update the column containing alphanumeric Course Codes to uppercase.

- SQLite is case insensitive and our Course Number attribute is an alphanumeric primary key. So SQLite perceives “CS500”, “cs500” and “cS500” etc. differently. If we use a check constraint while declaration that will mean, that the user is required to entered it in uppercase, or use the application logic.
- By using a very basic trigger after every INSERT in the course, we can fix this and enforce the constraint on this domain.
- The trigger will make sure that there are no duplicate values for the same course name and also help user input without any case constraints.

```
CREATE TRIGGER if not EXISTS update_course_title_to_uppercase
AFTER INSERT ON Course
FOR EACH ROW
BEGIN

UPDATE Course

SET

number = UPPER(number);

END;
```

## Screenshot of operation: Cs1014 and cs1015 converted to uppercase:

The screenshot shows a SQLite IDE interface. The top toolbar includes buttons for 'New Database', 'Open Database', 'Write Changes', 'Revert Changes', 'Open Project', 'Save Project', and 'Attach'. Below the toolbar are tabs for 'Database Structure', 'Browse Data', 'Edit Pragmas', and 'Execute SQL'. The main editor window displays the following SQL code:

```
2 CREATE TRIGGER if not EXISTS update_course_title_to_uppercase
3 AFTER INSERT ON Course
4 FOR EACH ROW
5 BEGIN
6
7 UPDATE Course
8
9 SET
10
11 number = UPPER(number);
12
13 END;
14
15 INSERT into Course(number,title,course_length) VALUES ('Cs1014','Cyber Warfare',12),('cs1015','Statistical Modelling',30);
16
17
18 SELECT * FROM Course;
```

Below the editor, there is a 'Find in editor' search bar with options for 'Case Sensitive', 'Whole Words', and 'Regular Expression'. The results of the SQL execution are displayed in a table with the following columns: 'number', 'title', and 'course\_length'.

|     | number  | title                   | course_length |
|-----|---------|-------------------------|---------------|
| 99  | CS10800 | ipsum porta elit,       | 49            |
| 100 | CS10900 | tellus.                 | 50            |
| 101 | CS1213  | Computer Systems        | 12            |
| 102 | CS1313  | Computer Tech           | 12            |
| 103 | CS1011  | Intergalactic SpaceWars | 12            |
| 104 | CS1012  | Mathematical Modelling  | 30            |
| 105 | CS1014  | Cyber Warfare           | 12            |
| 106 | CS1015  | Statistical Modelling   | 30            |

At the bottom, a status bar indicates: 'Execution finished without errors. Result: 106 rows returned in 26ms. At line 18: SELECT \* FROM Course;'.