SQLITE - TRIGGERS

SQLite **Triggers** are database callback functions, which are automatically performed/invoked when a specified database event occurs. Following are the important points about SQLite triggers:

- SQLite trigger may be specified to fire whenever a DELETE, INSERT or UPDATE of a particular database table occurs or whenever an UPDATE occurs on on one or more specified columns of a table.
- At this time, SQLite supports only FOR EACH ROW triggers, not FOR EACH STATEMENT triggers. Hence explicitly specifying FOR EACH ROW is optional.
- Both the WHEN clause and the trigger actions may access elements of the row being inserted, deleted or updated using references of the form **NEW.column-name** and **OLD.column-name**, where column-name is the name of a column from the table that the trigger is associated with.
- If a WHEN clause is supplied, the SQL statements specified are only executed for rows for which the WHEN clause is true. If no WHEN clause is supplied, the SQL statements are executed for all rows.
- The BEFORE or AFTER keyword determines when the trigger actions will be executed relative to the insertion, modification or removal of the associated row.
- Triggers are automatically dropped when the table that they are associated with is dropped.
- The table to be modified must exist in the same database as the table or view to which the trigger is attached and one must use just **tablename** not **database.tablename**.
- A special SQL function RAISE may be used within a trigger-program to raise an exception.

Syntax:

The basic syntax of creating a **trigger** is as follows:

```
CREATE  TRIGGER trigger_name [BEFORE|AFTER] event_name
ON table_name
BEGIN
 -- Trigger logic goes here....
END;
```

Here, **event_name** could be *INSERT, DELETE,* and *UPDATE* database operation on the mentioned table **table_name**. You can optionally specify FOR EACH ROW after table name.

Following is the syntax of creating a trigger on an UPDATE operation on one or more specified columns of a table as follows:

```
CREATE  TRIGGER trigger_name [BEFORE|AFTER] UPDATE OF column_name
ON table_name
BEGIN
 -- Trigger logic goes here....
END;
```

Example

Let us consider a case where we want to keep audit trial for every record being inserted in COMPANY table which we create newly as follows *DropCOMPANYtableifyoualreadyhaveit*:

```
sqlite> CREATE TABLE COMPANY(
   ID INT PRIMARY KEY     NOT NULL,
   NAME           TEXT    NOT NULL,
   AGE            INT     NOT NULL,
   ADDRESS        CHAR(50),
   SALARY         REAL
);
```

To keep audit trial, we will create a new table called AUDIT where log messages will be inserted whenever there is an entry in COMPANY table for a new record:

```
sqlite> CREATE TABLE AUDIT(
    EMP_ID INT NOT NULL,
    ENTRY_DATE TEXT NOT NULL
);
```

Here, ID is the AUDIT record ID, and EMP_ID is the ID which will come from COMPANY table and DATE will keep timestamp when the record will be created in COMPANY table. So now let's create a trigger on COMPANY table as follows:

```
sqlite> CREATE TRIGGER audit_log AFTER INSERT
ON COMPANY
BEGIN
   INSERT INTO AUDIT(EMP_ID, ENTRY_DATE) VALUES (new.ID, datetime('now'));
END;
```

Now, we will start actual work, let's start inserting record in COMPANY table which should result in creating an audit log record in AUDIT table. So let's create one record in COMPANY table as follows:

```
sqlite> INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (1, 'Paul', 32, 'California', 20000.00 );
```

This will create one record in COMPANY table, which is as follows:

```
ID          NAME        AGE         ADDRESS     SALARY
----------  ----------  ----------  ----------  ----------
1           Paul        32          California  20000.0
```

Same time, one record will be create in AUDIT table. This record is the result of a trigger, which we have created on INSERT operation on COMPANY table. Similar way you can create your triggers on UPDATE and DELETE operations based on your requirements.

```
EMP_ID      ENTRY_DATE
----------  -------------------
1           2013-04-05 06:26:00
```

Listing TRIGGERS

You can list down all the triggers from **sqlite_master** table as follows:

```
sqlite> SELECT name FROM sqlite_master
WHERE type = 'trigger';
```

Above SQLite statement will list down only one entry as follows:

```
name
----------
audit_log
```

If you want to list down triggers on a particular table, then use AND clause with table name as follows:

```
sqlite> SELECT name FROM sqlite_master
WHERE type = 'trigger' AND tbl_name = 'COMPANY';
```

Above SQLite statement will also list down only one entry as follows:

```
name
----------
audit_log
```

Dropping TRIGGERS

Following is the DROP command, which can be used to drop an existing trigger:

```
sqlite> DROP TRIGGER trigger_name;
```