Triggers In SQL

A `trigger` is a stored procedure in database which is automatically invoked whenever any special event occurs in the database. The event can be any event including INSERT, UPDATE and DELETE.

For eg: If you want to perfom a task after a record is inserted into the table then we can make use of `triggers`

Syntax for creating triggers

```
create trigger [trigger_name]
[before | after]
{insert | update | delete}
on [table_name]
[for each row | for each column]
[trigger_body]
```

`create trigger [trigger_name]` : Creates or replaces an existing trigger with the trigger_name.

`[before | after]` : Now we can specify when our trigger will get fired. It can be before updating the database or after updating the database.

Generally , `before` triggers are used to validate the data before storing it into the database.

`{insert | update | delete}` : Now, we specify the `DML operation` for which our trigger should get fired .

`on [table_name]` : Here, we specify the name of the table which is associated with the trigger.

`[for each row]` : This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected.

`[for each column]` : This specifies a column-level trigger, i.e., the trigger will be executed after the specified column is affected.

`[trigger_body`] : Here, we specify the operations to be performed once the trigger is fired.

Show Trigger

If you want to see all the triggers that are present in your database.

```
show triggers in database_name;
```

Drop Trigger

if you no longer want your trigger then you may delete it.

```
drop trigger trigger_name;
```

Example :

Let us consider we have our database named `library`. Consider a scenario where we want a trigger which is fired everytime any particular book is inserted into the `books` table . The `trigger` should add the logs of all the books that are inserted into the `books` table.

We have created two tables :

1. `books` : It will store all the books available in the library
2. bookrecord : It will generate a statement a log for the inserted book

```
Select * from library.books;
```

```
+----------+--------------+
| book_id  | book_name     |
+----------+--------------+
|          |                    |
|          |                    |
+----------+--------------+
```

Here, `book_id` is an auto-incremental field.

```
Select * from library.bookrecord;
```

```
+----------+--------------+-----------+
| SRNO     | bookid       | statement |
+----------+--------------+-----------+
|          |              |           |
|          |              |           |
+----------+--------------+-----------+
```

Here, `SRNO` is an auto-incremental field.

Now, we will create our trigger on the `books` table

```
create trigger library.addstatement
after insert
on library.books
for each row
insert into library.bookrecord(bookid,statement) values
(NEW.book_id,concat('New book named ',NEW.book_name,"  added
at ",curdate()));
```

In MySQL, `NEW` is used to access the currently inserted row. We are inserting the log for the currently inserted book in our database.

Now we will insert a book and wait for the output.

```
insert into library.books(book_name) values ("Harry Potter and
the Goblet of fire");
```

Output for `books`:

```
+----------+---------------------------------------------+
| book_id  | book_name                                   |
+----------+---------------------------------------------+
|    1     |         Harry Potter and the Goblet of fire
|
|          |
|
+----------+---------------------------------------------+
```

Output for `bookrecord`:

```
+----------+--------------+----------------------------------
-------------------------------------------------+
| SRNO     | bookid       | statement
|
+----------+--------------+----------------------------------
-------------------------------------------------+
|   1      |      1       |     New book named Harry Potter
and the Goblet of fire  added at 2021-10-22      |
|          |              |
|
+----------+--------------+----------------------------------
-------------------------------------------------+
```

See. it worked!!

Conclusion:

Here, you learnt what are triggers and how you create them. You can create different types of triggers based on your needs and requirements.

`Transaction Control Language`

- `Transaction Control Language` can be defined as the portion of a database language used for `maintaining consistency` of the database and `managing transactions` in the database.

- A set of `SQL statements` that are `co-related logically and executed on the data stored in the table` is known as a `transaction`.

`TCL` Commands

- `COMMIT` Command
- `ROLLBACK` Command
- `SAVEPOINT` Command

`COMMIT`

The main use of `COMMIT` command is to `make the transaction permanent`. If there is a need for any transaction to be done in the database that transaction permanent through commit command.

Syntax

```
COMMIT;
```

`ROLLBACK`

Using this command, the database can be `restored to the last committed state`. Additionally, it is also used with savepoint command for jumping to a savepoint in a transaction.

Syntax

```
ROLLBACK TO savepoint-name;
```

`SAVEPOINT`

The main use of the Savepoint command is to save a transaction temporarily. This way users can rollback to the point whenever it is needed.

Syntax

```
SAVEPOINT savepoint-name;
```

Examples

This is purchase table that we are going to use through this tutorial

item	price	customer_name
Pen	10	Sanskriti
Bag	1000	Sanskriti
Vegetables	500	Sanskriti
Shoes	5000	Sanskriti
Water Bottle	800	XYZ
Mouse	120	ABC
Sun Glasses	1350	ABC

```
UPDATE purchase SET price = 20 WHERE item = "Pen";
```

O/P : Query OK, 1 row affected (3.02 sec) (Update the price of Pen set it from 10 to 20)

```
SELECT * FROM purchase;
```

O/P

item	price	customer_name
Pen	20	Sanskriti
Bag	1000	Sanskriti
Vegetables	500	Sanskriti
Shoes	5000	Sanskriti
Water Bottle	800	XYZ
Mouse	120	ABC
Sun Glasses	1350	ABC

```
START TRANSACTION;
```

Start transaction

```
COMMIT;
```

Saved/ Confirmed the transactions till this point

```
ROLLBACK;
```

Lets consider we tried to rollback above transaction

```
SELECT * FROM purchase;
```

O/P:

item	price	customer_name
Pen	20	Sanskriti
Bag	1000	Sanskriti
Vegetables	500	Sanskriti
Shoes	5000	Sanskriti
Water Bottle	800	XYZ
Mouse	120	ABC
Sun Glasses	1350	ABC

As we have committed the transactions the `rollback` will not affect anything

```
SAVEPOINT  sv_update;
```

Create the `savepoint` the transactions above this will not be rollbacked

```
UPDATE purchase SET price = 30 WHERE item = "Pen";
```

O/P : Query OK, 1 row affected (0.57 sec)

Rows matched: 1 Changed: 1 Warnings: 0

```
SELECT * FROM purchase;
```

item	price	customer_name
Pen	30	Sanskriti
Bag	1000	Sanskriti
Vegetables	500	Sanskriti
Shoes	5000	Sanskriti
Water Bottle	800	XYZ
Mouse	120	ABC
Sun Glasses	1350	ABC

price of pen is changed to 30 using the `update` command

```
ROLLBACK to sv_update;
```

Now if we `rollback` to the `savepoint` price should be 20 after `rollback` lets see

```
SELECT * FROM purchase;
```

item	price	customer_name
Pen	20	Sanskriti

item	price	customer_name
Bag	1000	Sanskriti
Vegetables	500	Sanskriti
Shoes	5000	Sanskriti
Water Bottle	800	XYZ
Mouse	120	ABC
Sun Glasses	1350	ABC
Torch	850	ABC

As expected we can see `update` query is rollbacked to sv_update.

Conclusion

With this short tutorial we have learnt TCL commands.

Data Control Language

DCL commands are used to grant and take back authority from any database user.

`DCL` Commands

- `GRANT` Command
- `REVOKE` Command

GRANT

GRANT is used to give user access privileges to a database.

Syntax

```
GRANT privilege_name ON objectname TO user;
```

REVOKE

REVOKE remove a privilege from a user. REVOKE helps the owner to cancel previously granted permissions.

Syntax

```
REVOKE privilege_name ON objectname FROM user;
```

DCL Examples

```
SELECT * FROM purchase;
```

Output:

```
| item         | price | customer_name |
|--------------|-------|---------------|
| Pen          |    20 | Sanskriti     |
| Bag          |  1000 | Sanskriti     |
| Vegetables   |   500 | Sanskriti     |
| Shoes        |  5000 | Sanskriti     |
| Water Bottle |   800 | XYZ           |
| Mouse        |   120 | ABC           |
| Sun Glasses  |  1350 | ABC           |
| Torch        |   850 | ABC           |
```

- Lets start with GRANT command:

```
  GRANT INSERT ON purchase TO 'Sanskriti'@'localhost';
```

Output:

```
#### O/P Query OK, 0 rows affected (0.31 sec)
```

Description In above command we have granted user Sanskriti priviledge to `Insert` into purchase table.

- Now if I login as Sanskriti and try to run `Select` statement as given below what should happen?

```
SELECT * FROM purchase;
```

Output:

```
#### O/P ERROR 1142 (42000): SELECT command denied to user
'Sanskriti'@'localhost' for table 'purchase'
```

Yup as expected it gives error because we have granted insert operation to Sanskriti.

- So lets try inserting data to purchase table:

```
INSERT INTO purchase values("Laptop", 100000, "Sanskriti");
```

Output:

```
#### O/P Query OK, 1 row affected (0.34 sec)
```

Yes! It works!

- Now I am checking the purchase table from my original account:

```
SELECT * FROM purchase;
```

Output:

```
| item         | price  | customer_name |
|-------------|--------|---------------|
| Pen          |     20 | Sanskriti     |
| Bag          |   1000 | Sanskriti     |
| Vegetables   |    500 | Sanskriti     |
| Shoes        |   5000 | Sanskriti     |
| Water Bottle |    800 | XYZ           |
| Mouse        |    120 | ABC           |
| Sun Glasses  |   1350 | ABC           |
| Torch        |    850 | ABC           |
| Laptop       | 100000 | Sanskriti     |
```

As you can see, the row is inserted.

- Now lets try `Revoke` command:

```
REVOKE INSERT ON purchase FROM 'Sanskriti'@'localhost';
```

Output:

```
#### O/P Query OK, 0 rows affected (0.35 sec)
```

Now we have revoked the insert priviledge from Sanskriti.

- If Sanskriti runs insert statement it should give error:

```
INSERT INTO purchase values("Laptop", 100000, "Sanskriti");
```

Output:

```
#### O/P ERROR 1142 (42000): INSERT command denied to user
'Sanskriti'@'localhost' for table 'purchase'
```

Conclusion

Through this tutorial we have learnt `DCL` commands and their usage.