SQL_Triggers

April 23, 2023

This notebook will contain all neccessary code and explanation for sql triggers

SQL trigger has three primary events , insert , update and delete , with the help of triggers , whenever we are trying to make any of the modification in our main table with insert/update/delete , based on our given specific condition the sql query runs and our data gets stored accordingly , as an example , we can set a trigger that states if any data gets deleted from main table then the deleted data will be stored into a backup table created by us , or let's say we have given a condition that during data insertion if the salary value is somehow inserted as negative number then insert 0 insetad of the negative numbers.

```
-   Trigger time frame - before/after
-   Trigger events - insert/update/delete
-   trigger order - follows/preceeds
```

Lets go ahead and create a main and a back up table first here.

```
-    use practisedb
-    show tables
-    create table main (id int ,salary int);
-    create table back_up (id int ,salary int);
```

now let's add values in the main table

```
-   INSERT INTO main VALUES ('1','20000'),('2','10000'),
    ('3','25000'),('4','15000');
```

Before Insert Trigger Lets first create a before insert trigger , our condition here will be that during data insertion if the salary value is somehow inserted as negative number then insert 0 insetad of the negative numbers.

```
-   Delimiter //
    create trigger salary_check
    before insert on main
    for each row
    begin
    if new.salary < 0 then set new.salary = 0 ;
    end if;
    end //
```

Now let's try to insert some data with negative values -

```
-   insert into main values (6,14000) , (7,-34566);
```

But now when we will check our table , we will see that the negative value got added as 0 only in the data.

```
-   SELECT * FROM practisedb.main;
```

After Insert Trigger Let's understand after insert trigger now. Suppose the data that's being inserted into the table contains null value for a specific column in it and we somehow require a value to be added in it. One way to do it is by just setting a not null constraint , but what if we don't do that. In such scenarios we will use an after insert trigger , with this trigger , everytime a null value gets inserted into that specified column a message will be generated into a separate table , let's call it messages where it will display that these records are null and values needs to be filled in.

```
-   create table customers
    (id int auto_increment primary key ,
        name varchar(40) not null , dob date);
-   create table message
    (id int auto_increment ,
            messageid int ,
            message varchar(400) not null ,
    primary key(id,messageid));
```

Now let's go ahead and create our trigger

```
•   Delimiter //
    create trigger
    check_null_dob
    after insert
    on customers for each row
    begin
    if new.dob is null then
    insert into message (messageid,message)
    values (new.id , concat('Hi ',new.name,' , please update your date of birth' ));
    end if;
    end //
```

Now let's add some data in the customers table as below -

```
-   INSERT INTO `practisedb`.`customers` (`id`, `name`, `dob`) VALUES ('1', 'joy', '1988-01-11
-   INSERT INTO `practisedb`.`customers` (`id`, `name`, `dob`) VALUES ('2', 'harry', Null);
```

Now when we will check the message table , we will see that for the record where name is harry , a message is displayed requesting to update their DOB.

Before Update Trigger This is simply updating a range of records with the help of a trigger based on given condition. Let's create an employee table where we will perform mass salary updation with 'before update trigger'.

```
-   create table employees (id int primary key , name varchar(40) not null , salary int not nu
-   insert into employees values (1,'harry',15000),(2,'barry',10000),(3,'larry',19000),(4,'ca
```

And the trigger code will be like as ,

```
-   delimiter //
    create trigger salary_update
    before update
    on employees
    for each row
    begin
    if new.salary = 10000 then set new.salary = 15000 ;
    elseif new.salary < 10000 then set new.salary = 5000 ;
    end if;
    end //
```

now if we use the update syntax as below -

```
-   update employees set salary = 8000;
```

We will get all salary output as 5000 as mentioned in the trigger.

After Update Trigger In here after every update the old values that were updated into new ones , goes into a back up table and stays stored there so that old updated values can be recalled.

Let's create a back up table for the after update trigger execution

```
-   create table update_back_up (id int , salary int);
```

Now let's create the trigger

```
-   Delimiter //
    create trigger trg25
    after update
    on main
    for each row
    begin
    insert into update_back_up values(old.id , old.salary);
    end //
```

Now let's perform an update on the main table as below -

```
-   update main set salary = 21345 where id =1;
```

Now if we check the back up table we have just created , we would be able to see the old data of id=1 that we just updated is safely stored in the back up table.

Before Delete In here we will be able to restrict our data to be deleted using a error/warning message before the delete happens. On the below trigger emp_data table is used , and sqlstate 45000 means unhandled user exception.

```
-   Delimiter //
    create trigger trg10
    before delete
    on emp_data
    for each row
    begin
    signal sqlstate '45000' set message_text = "not allowed";
    end //
```

Now deleteing a record to see the error message we receive

```
-   delete from emp_data where id = 1;
```

We will see the error message popping up as described above.

After Delete In After Delete Trigger the record gets deleted from the main table but at the same time it also gets stored in a back up table. let's create a main and a back up table with same columns and add some values in the main table first.

```
-   create table main (id int , salary int);
    create table back_up (id int , salary int);
    insert into main values (1,12000),(2,25000),(3,26000),(4,19000);
```

Now let's create the trigger

```
-   Delimiter //
    create trigger trg18
    after delete
    on main
    for each row
    begin
    insert into back_up values (old.id,old.salary);
    end //
```

Now let's delete a data from main table

```
-   delete from main where id = 3;
```

If we check the back up table now , the data will reflect in there , the exact data , that's deleted.

Showcase all the triggers

```
-   show triggers from practisedb;
```

Delete trigger

```
-   drop trigger trg1 -( trigger name )
```