WEEK 8

MORE SQL - TRIGGERS

STUDENT OBJECTIVES

- Upon completion of this video, you should be able to:
  - List 3 reasons why triggers are useful
  - Identity the parts of a trigger
  - Trace SQL statements that use a trigger and determine what will be in the resulting tables

TRIGGERS AND STORED PROCEDURES

- Triggers are a set of SQL statements that execute when a certain event occurs in a table
- Similar to a constraint but more powerful:
  - Example: a constraint will disallow a salary to go above a certain amount
  - A trigger can check how much the salary changes and if the change is above a certain amount, it can cause a user-defined function to notify an administrator about the change.
- Almost all database management systems have triggers
- Can help with the following:
  - If a business rule changes, you just need to change trigger not code
  - Improved performance (rules run on the server)
  - Global enforcement of business rules

SAMPLE MYSQL TRIGGER

```
9
delimiter //                    1          2       3          4
CREATE TRIGGER upd_check BEFORE UPDATE ON account
6 FOR EACH ROW
 8BEGIN             5
   7IF NEW.amount < 0 THEN
        SET NEW.amount = 0;
    ELSEIF NEW.amount > 100 THEN
        SET NEW.amount = 100;
    END IF;
  END;//
9 delimiter ;
```

	Parts of the trigger
1	Trigger Name (upd_check)
2	Trigger activation time (BEFORE)
3	Triggering event (UPDATE)
4	Triggering table name (account)
5	Attribute in table (NEW.amount)
6	Granularity (FOR EACH ROW)
7	Trigger condition (IF NEW.amount < 0 THEN)
8	Trigger body (BEGIN … END;)
9	In MySQL need to change the delimiter temporarily.

QUESTION: What will cause the trigger to be activated?

In the previous slide, any time a row in the account table is updated

When you no longer need the trigger, do the following command:

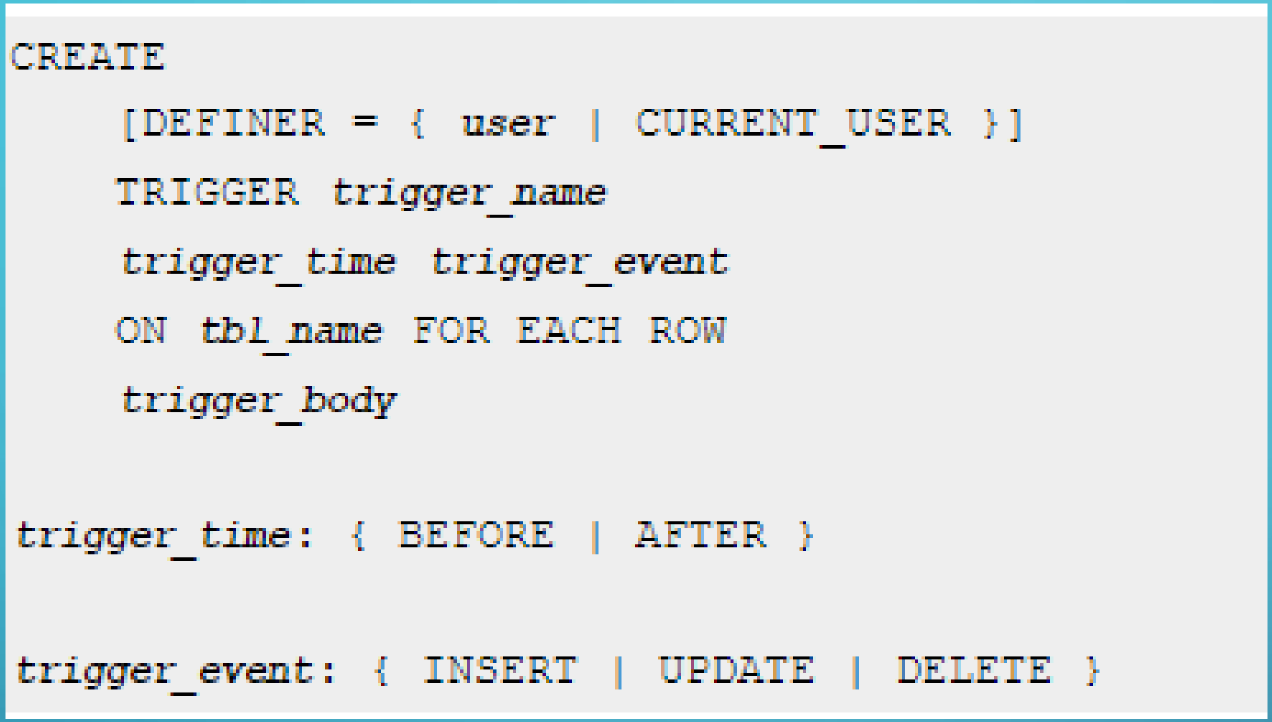
DROP TRIGGER [IF EXISTS] trigger_name

QUESTION: What do you think happens if have a table called *account* and it has a trigger associated with it and you do the command:

DROP TABLE account?

All triggers associated with that table will be dropped.

Parts of the trigger in more detail:



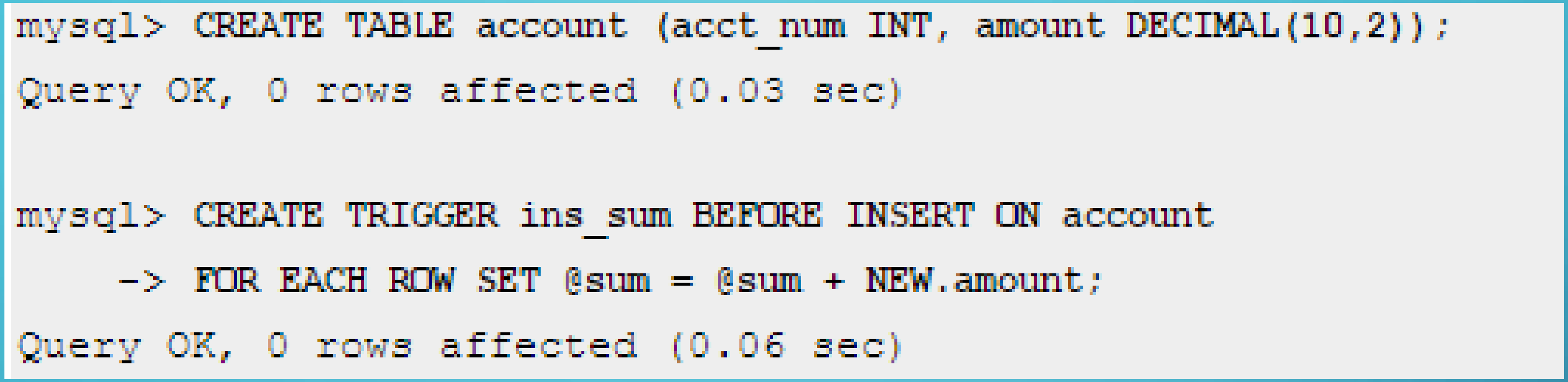
```
CREATE
    [DEFINER = { user | CURRENT_USER }]
    TRIGGER trigger_name
    trigger_time trigger_event
    ON tbl_name FOR EACH ROW
    trigger_body

trigger_time: { BEFORE | AFTER }

trigger_event: { INSERT | UPDATE | DELETE }
```

NOTES:

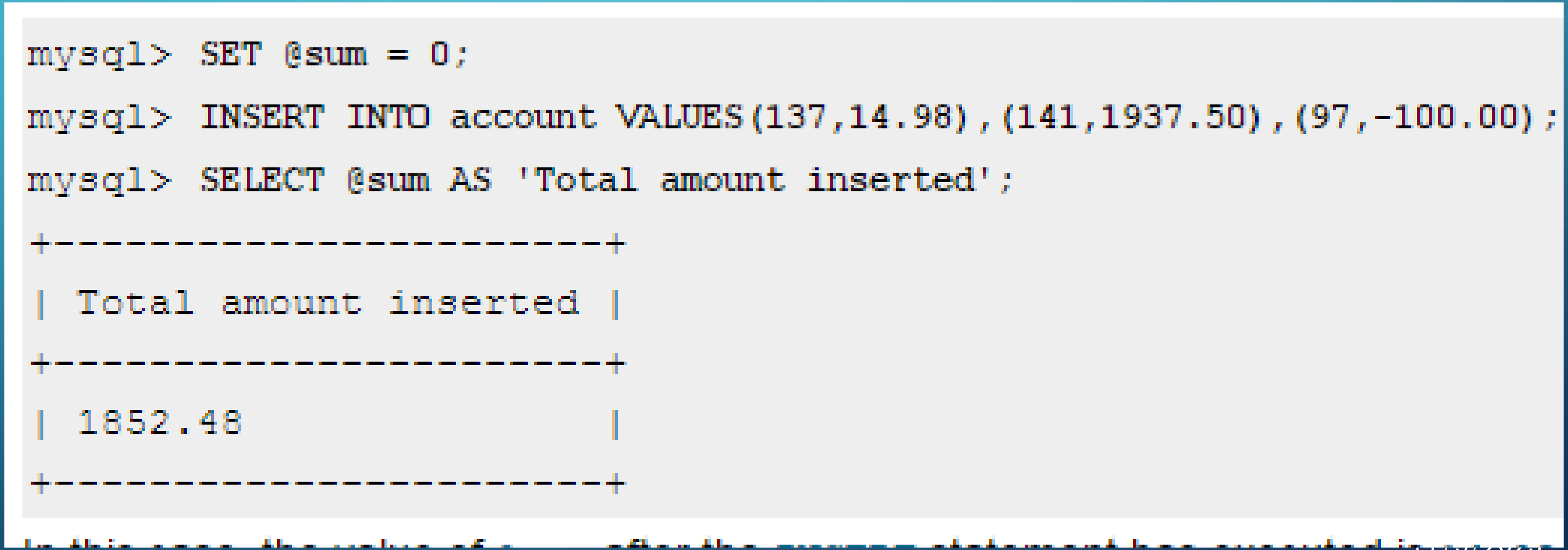
- you **cannot** have multiple triggers for a given table that have the same trigger event and action time, e.g. you can't have 2 BEFORE INSERT triggers BUT you can have a BEFORE and AFTER trigger or a BEFORE INSERT and BEFORE UPDATE triggers

- ***Granularity:***
  - FOR EACH ROW: the trigger is activated every time a row is changed
- ***Trigger Condition:***
  - similar to the WHERE clause in an SQL statement.
  - If you do not have a trigger condition, the trigger's body executes every time the trigger is activated.

ANOTHER EXAMPLE:



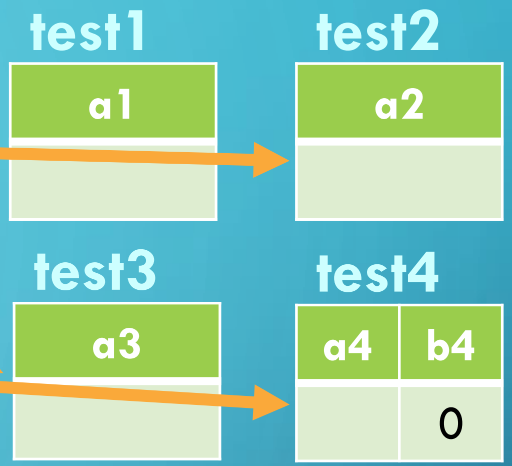
```
mysql> CREATE TABLE account (acct_num INT, amount DECIMAL(10,2));
Query OK, 0 rows affected (0.03 sec)

mysql> CREATE TRIGGER ins_sum BEFORE INSERT ON account
    -> FOR EACH ROW SET @sum = @sum + NEW.amount;
Query OK, 0 rows affected (0.06 sec)
```



```
mysql> SET @sum = 0;
mysql> INSERT INTO account VALUES(137,14.98),(141,1937.50),(97,-100.00);
mysql> SELECT @sum AS 'Total amount inserted';
+-----------------------+
| Total amount inserted |
+-----------------------+
| 1852.48               |
+-----------------------+
```

ANOTHER EXAMPLE:

```
CREATE TABLE test1(a1 INT);
CREATE TABLE test2(a2 INT);
CREATE TABLE test3(a3 INT NOT NULL AUTO_INCREMENT PRIMARY KEY);
CREATE TABLE test4(
  a4 INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  b4 INT DEFAULT 0
);
delimiter |
CREATE TRIGGER testref BEFORE INSERT ON test1
  FOR EACH ROW
  BEGIN
    INSERT INTO test2 SET a2 = NEW.a1;
    DELETE FROM test3 WHERE a3 = NEW.a1;
    UPDATE test4 SET b4 = b4 + 1 WHERE a4 = NEW.a1;
  END;
|
delimiter ;
INSERT INTO test3 (a3) VALUES (NULL), (NULL), (NULL), (NULL), (NULL), (NULL), (NULL),
(NULL), (NULL), (NULL);

INSERT INTO test4 (a4) VALUES (0), (0), (0), (0), (0), (0), (0), (0), (0), (0);
INSERT INTO test1 VALUES (1), (3), (1), (7), (1), (8), (4), (4);
SELECT * from test1;
SELECT * from test2;
SELECT * from test3;
SELECT * from test4;
```



ANOTHER EXAMPLE CONTINUED:

- Now try this out in your Virtual Machine

```
vetoffice=> INSERT INTO test3 (b3) VALUES ('cow'), ('cow'), ('cow'), ('cow'), ('cow'), ('cow'),
('cow'), ('cow'), ('cow'), ('cow');
INSERT 0 10
vetoffice=> SELECT * FROM test3;
 a3 | b3
----+-----
  1 | cow
  2 | cow
  3 | cow
  4 | cow
  5 | cow
  6 | cow
  7 | cow
  8 | cow
  9 | cow
 10 | cow
(10 rows)

vetoffice=> INSERT INTO test4 (b4) VALUES (0), (0), (0), (0), (0), (0), (0), (0), (0), (0);
INSERT 0 10
vetoffice=> SELECT * FROM test4;
 a4 | b4
----+----
  1 |  0
  2 |  0
  3 |  0
  4 |  0
  5 |  0
  6 |  0
  7 |  0
  8 |  0
  9 |  0
 10 |  0
(10 rows)
```

```
vetoffice=> INSERT INTO test1  VALUES (1), (3), (1), (7), (1), (8), (4), (4);
INSERT 0 8
vetoffice=> SELECT * from test1;
 a1
----
  1
  3
  1
  7
  1
  8
  4
  4
(8 rows)

vetoffice=> SELECT * from test2;
 a2
----
  1
  3
  1
  7
  1
  8
  4
  4
(8 rows)
```

```
vetoffice=> SELECT * from test3;
 a3 | b3
----+-----
  2 | cow
  5 | cow
  6 | cow
  9 | cow
 10 | cow
(5 rows)

vetoffice=> SELECT * from test4;
 a4 | b4
----+----
  2 |  0
  5 |  0
  6 |  0
  9 |  0
 10 |  0
  3 |  1
  7 |  1
  1 |  3
  8 |  1
  4 |  2
(10 rows)
```

QUESTION: Assume we have the tables:

- STUDENT(Student_Num, LastName, …Age)
- UNIV_STATS(Num_of_Students, Total_Age, Average_Age, …)

Write a trigger that will keep the UNIV_STATS table accurate: