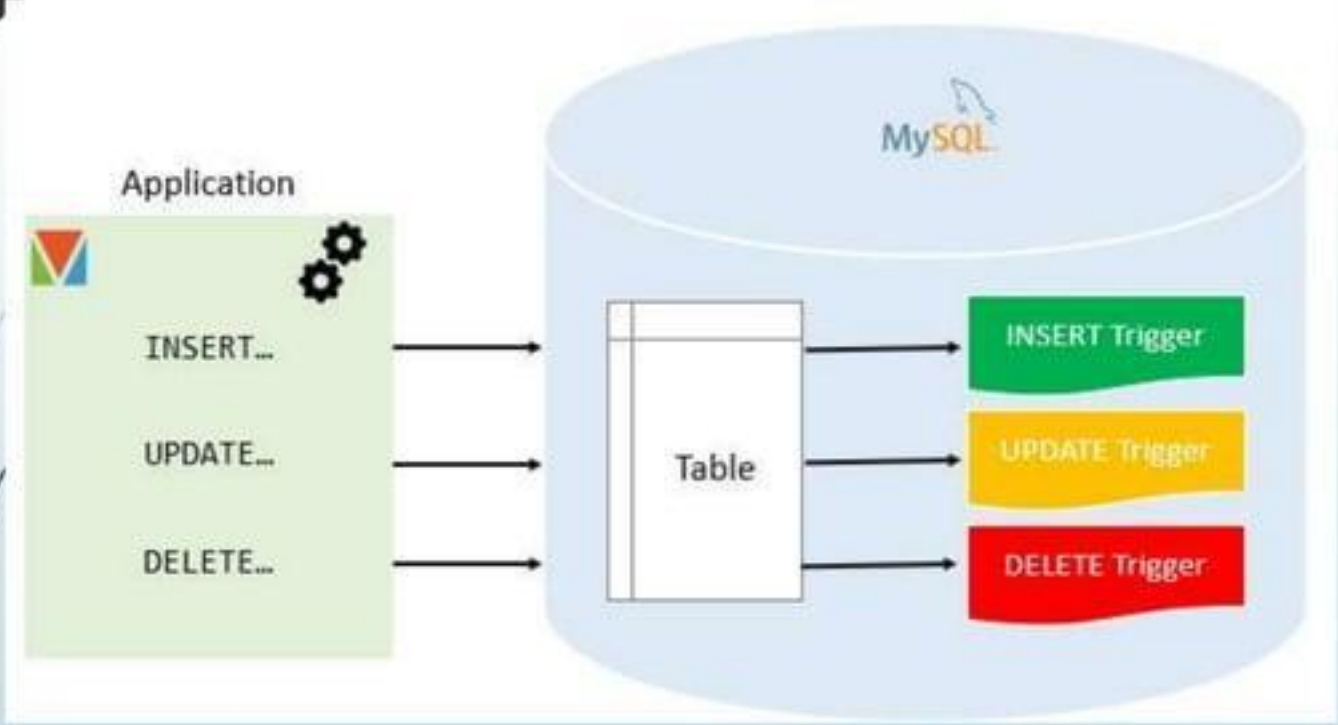
MYSQL Triggers

- In MySQL, a trigger is a stored program invoked automatically in response to an event such as insert, update, or delete that occurs in the associated table.
- To monitor a database and take a corrective action when a condition occurs
- For example, you can define a trigger that is invoked automatically before a new row is inserted into a table.
- MySQL supports triggers that are invoked in response to the INSERT, UPDATE or DELETE event.

Types of triggers: Row-level triggers and Statement-level triggers:

- A **row-level** trigger is activated for each row that is inserted, updated, or deleted.
- For example, if a table has 100 rows inserted, updated, or deleted, the trigger is automatically invoked 100 times for the 100 rows affected.
- A **statement-level** trigger is executed once for each transaction regardless of how many rows are inserted, updated, or deleted.
- MySQL supports only row-level triggers. It doesn't support statement-level triggers.


Application
INSERT...
UPDATE...
DELETE...
MySQL
Table
INSERT Trigger
UPDATE Trigger
DELETE Trigger

MYSQL Triggers syntax

```
CREATE TRIGGER trigger-name
trigger-time trigger-event
ON table-name
FOR EACH ROW
   trigger-action;
```

trigger-time ∈ {BEFORE, AFTER}

trigger-event ∈ {INSERT,DELETE,UPDATE}

Triggers

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

trigger_time: { BEFORE | AFTER }

trigger_event: { INSERT | UPDATE | DELETE }

trigger_order: { FOLLOWS | PRECEDES } other_trigger_name
```

SQL Triggers: An Example

- We want to create a trigger to update the total salary of a department when a new employee is hired

```
mysql> select * from employee;
+----+------+---------+--------+------------+------+
| id | name | superid | salary | bdate      | dno  |
+----+------+---------+--------+------------+------+
|  1 | john |       3 | 100000 | 1960-01-01 |    1 |
|  2 | mary |       3 |  50000 | 1964-12-01 |    3 |
|  3 | bob  |    NULL |  80000 | 1974-02-07 |    3 |
|  4 | tom  |       1 |  50000 | 1970-01-17 |    2 |
|  5 | bill |    NULL |   NULL | 1985-01-20 |    1 |
+----+------+---------+--------+------------+------+
5 rows in set (0.00 sec)

mysql> select * from deptsal;
+---------+-------------+
| dnumber | totalsalary |
+---------+-------------+
|       1 |      100000 |
|       2 |       50000 |
|       3 |      130000 |
+---------+-------------+
3 rows in set (0.00 sec)
```

SQL Triggers: Another Example

- Create a trigger to update the total salary of a department when a new employee is hired:

```
mysql> delimiter !
mysql> create trigger update_salary
    -> after insert on employee
    -> for each row
    -> begin
    ->       if new.dno is not null then
    ->           update deptsal
    ->           set totalsalary = totalsalary + new.salary
    ->           where dnumber = new.dno;
    ->       end if;
    -> end !
Query OK, 0 rows affected (0.06 sec)

mysql> delimiter ;
```

- The keyword “new” refers to the new row inserted

SQL Triggers: Another Example – Part 2

```
mysql> select * from deptsal;
+---------+--------------+
| dnumber | totalsalary  |
+---------+--------------+
|       1 |       100000 |
|       2 |        50000 |
|       3 |       130000 |
+---------+--------------+
3 rows in set (0.00 sec)

mysql> insert into employee values (6,'lucy',null,90000,'1981-01-01',1);
Query OK, 1 row affected (0.08 sec)

mysql> select * from deptsal;
+---------+--------------+
| dnumber | totalsalary  |
+---------+--------------+
|       1 |       190000 |
|       2 |        50000 |
|       3 |       130000 |
+---------+--------------+
3 rows in set (0.00 sec)

mysql> insert into employee values (7,'george',null,45000,'1971-11-11',null);
Query OK, 1 row affected (0.02 sec)

mysql> select * from deptsal;
+---------+--------------+
| dnumber | totalsalary  |
+---------+--------------+
|       1 |       190000 |
|       2 |        50000 |
|       3 |       130000 |
+---------+--------------+
3 rows in set (0.00 sec)

mysql> drop trigger update_salary;
Query OK, 0 rows affected (0.00 sec)
```

totalsalary increases by 90K

totalsalary did not change

NEW and OLD: Extension to MYSQL triggers

There are two MySQL extension to triggers '**OLD**' and '**NEW**'.

✓ OLD and NEW are not case sensitive.

•Within the trigger body, the OLD and NEW keywords enable you to access columns in the rows affected by a trigger

•In an INSERT trigger, only NEW.col_name can be used.

•In a UPDATE trigger, you can use OLD.col_name to refer to the columns of a row before it is updated and NEW.col_name to refer to the columns of the row after it is updated.

•In a DELETE trigger, only OLD.col_name can be used; there is no new row.

NEW and OLD keywords

- A column named with OLD is read only.
- You can refer to it (if you have the SELECT privilege), but not modify it.
- You can refer to a column named with NEW if you have the SELECT privilege for it.

  In a BEFORE trigger, you can also change its value with SET NEW.col_name = value if you have the UPDATE privilege for it.
- This means you can use a trigger to modify the values to be inserted into a new row or used to update a row. (Such a SET statement has no effect in an AFTER trigger because the row change will have already occurred.)

SQL Triggers: Another Example – Part 3

- A trigger to update the total salary of a department when an employee tuple is modified:

```
mysql> delimiter !
mysql> create trigger update_salary2
    -> after update on employee
    -> for each row
    -> begin
    ->       if old.dno is not null then
    ->          update deptsal
    ->          set totalsalary = totalsalary - old.salary
    ->          where dnumber = old.dno;
    ->       end if;
    ->       if new.dno is not null then
    ->          update deptsal
    ->          set totalsalary = totalsalary + new.salary
    ->          where dnumber = new.dno;
    ->       end if;
    -> end !
Query OK, 0 rows affected (0.06 sec)
```

SQL Triggers: An Example – Part 4

```
mysql> deliniter ;
mysql> select * from employee;
+----+---------+---------+--------+------------+------+
| id | name    | superid | salary | bdate      | dno  |
+----+---------+---------+--------+------------+------+
|  1 | john    |       3 | 100000 | 1960-01-01 |    1 |
|  2 | mary    |       3 |  50000 | 1964-12-01 |    3 |
|  3 | bob     |    NULL |  80000 | 1974-02-07 |    3 |
|  4 | tom     |       1 |  50000 | 1970-01-17 |    2 |
|  5 | bill    |    NULL |   NULL | 1985-01-20 |    1 |
|  6 | lucy    |    NULL |  90000 | 1981-01-01 |    1 |
|  7 | george  |    NULL |  45000 | 1971-11-11 | NULL |
+----+---------+---------+--------+------------+------+
7 rows in set (0.00 sec)

mysql> select * from deptsal;
+----------+-------------+
| dnumber  | totalsalary |
+----------+-------------+
|        1 |      190000 |
|        2 |       50000 |
|        3 |      130000 |
+----------+-------------+
3 rows in set (0.00 sec)

mysql> update employee set salary = 100000 where id = 6;
Query OK, 1 row affected (0.03 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from deptsal;
+----------+-------------+
| dnumber  | totalsalary |
+----------+-------------+
|        1 |      200000 |
|        2 |       50000 |
|        3 |      130000 |
+----------+-------------+
3 rows in set (0.00 sec)
```

SQL Triggers: Another Example – Part 5

- A trigger to update the total salary of a department when an employee tuple is deleted:

```
mysql> deliniter !
mysql> create trigger update_salary3
    -> before delete on employee
    -> for each row
    -> begin
    ->       if (old.dno is not null) then
    ->           update deptsal
    ->           set totalsalary = totalsalary - old.salary
    ->           where dnumber = old.dno;
    ->       end if;
    -> end !
Query OK, 0 rows affected (0.08 sec)

mysql> deliniter ;
```

SQL Triggers: Another Example – Part 6

```
mysql> select * from employee;
+----+--------+---------+--------+------------+------+
| id | name   | superid | salary | bdate      | dno  |
+----+--------+---------+--------+------------+------+
|  1 | john   |       3 | 100000 | 1960-01-01 |    1 |
|  2 | mary   |       3 |  50000 | 1964-12-01 |    3 |
|  3 | bob    |    NULL |  80000 | 1974-02-07 |    3 |
|  4 | tom    |       1 |  50000 | 1970-01-17 |    2 |
|  5 | bill   |    NULL |   NULL | 1985-01-20 |    1 |
|  6 | lucy   |    NULL | 100000 | 1981-01-01 |    1 |
|  7 | george |    NULL |  45000 | 1971-11-11 | NULL |
+----+--------+---------+--------+------------+------+
7 rows in set (0.00 sec)
```

```
mysql> select * from deptsal;
+----------+--------------+
| dnumber  | totalsalary  |
+----------+--------------+
|        1 |       200000 |
|        2 |        50000 |
|        3 |       130000 |
+----------+--------------+
3 rows in set (0.00 sec)
```

```
mysql> delete from employee where id = 6;
Query OK, 1 row affected (0.02 sec)

mysql> delete from employee where id = 7;
Query OK, 1 row affected (0.03 sec)

mysql> select * from deptsal;
+----------+--------------+
| dnumber  | totalsalary  |
+----------+--------------+
|        1 |       100000 |
|        2 |        50000 |
|        3 |       130000 |
+----------+--------------+
3 rows in set (0.00 sec)
```

A Few Things to Note

- A given trigger can only have one event.
- If you have the same or similar processing that has to go on during insert and delete, then it's best to have that in a procedure or function and then call it from the trigger.
- A good naming standard for a trigger is <table_name>_event if you have the room for that in the name.
- Just like a function or a procedure, the trigger body will need a begin ... end unless it is a single statement trigger.

The Special Powers of a Trigger

- While in the body of a trigger, there are potentially two sets of column values available to you, with special syntax for denoting them.
  - old.<column name> will give you the value of the column before the DML statement executed.
  - new.<column name> will give you the value of that column **after** the DML statement executed.
- Insert triggers have no old values available, and delete triggers have no new values available for obvious reasons. Only update triggers have both the old and the new values available.
- Only triggers can access these values this way.

More Examples

- Simplified example of a parent table: hospital_room as the parent and hospital_bed as the child.
- The room has a column: max_beds that dictates the maximum number of beds for that room.
- The hospital_bed table has a before insert trigger that checks to make sure that the hospital room does not already have its allotted number of beds.

The Trigger

```
CREATE DEFINER=`root`@`localhost`
TRIGGER `programming`.`hospital_bed_BEFORE_INSERT`
BEFORE INSERT ON `hospital_bed` FOR EACH ROW
BEGIN
    declare max_beds_per_room int;
    declare current_count int;
    select    max_beds into max_beds_per_room
    from      hospital_room
    where     hospital_room_no = new.room_id;
    select    count(*) into current_count
    from      hospital_bed
    where     room_id = new.room_id;
    if current_count >= max_beds_per_room then
        signal sqlstate '45000' set message_text='Too many beds in that room already!';
    end if;
END;
```

Firing the trigger

```
insert into hospital_bed (room_id, hospital_bed_id)
values    ('323B', 1);
insert into hospital_bed (room_id, hospital_bed_id)
values    ('323B', 2);
insert into hospital_bed (room_id, hospital_bed_id)
values    ('323B', 3);
insert into hospital_bed (room_id, hospital_bed_id)
values    ('323B', 4);
insert into hospital_bed (room_id, hospital_bed_id)
values    ('323B', 5);
```

Error Code: 1644. Too many beds in that room already!

Using a Stored Procedure Instead

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `too_many_beds`(in room_id varchar(45))
BEGIN
	declare max_beds_per_room int;
	declare current_count int;
	declare room_count int;
	-- see if the hospital room exists
	selectcount(*) into room_count
	from  hospital_room
	where	hospital_room_no = room_id;
	if room_count = 1 then -- we can see if room for 1 more bed
	begin
		selectmax_beds into max_beds_per_room
		from  hospital_room
		where	hospital_room_no = room_id;
		-- count the beds in this room
		selectcount(*) into current_count
		from  hospital_bed
		where	room_id = room_id;
		if current_count >= max_beds_per_room then
			-- flag an error to abort if necessary
			signal sqlstate '45000' set message_text='Too many beds in that room already!';
		end if;
	end;
	end if;
END
```

Comments on the Procedure

- Because that is in isolation from the beds table, we have to check to make sure that the room number is viable.
- As a stored procedure, this can be called directly from the command line as a means of unit testing.
- I'm still not too sure how exacting the typing of the parameters has to be. For instance, does that one argument have to be exactly a varchar(45) in order for it to work, or not?

Viewing Your Triggers

- MySQL has a schema that has tables for all of the information that is needed to define and run the data in the database. This is meta data.
- select * from information_schema.triggers where trigger_schema='<your schema name>'; -- retrieve the trigger information for the triggers in <your schema name>.
- Alternatively, you can use the "show triggers" command (this is not SQL) that will display a report of your triggers from the default schema.

```
mysql> show triggers;
```

Viewing Your Triggers (Continued)

- If you're using MySQL Workbench, the IDE provides access to your triggers:
  - In the navigator pane, right click the table that has the trigger.
  - Select "Alter Table"
  - This will open up a rather lavish dialog which has tabs down near the bottom. One of those tabs is "Triggers". Select that.
  - That will open up **another** dialog, and over to the left will be the list of events that you can define triggers for.
    - At this point, you can right click one of those events and it will pop up a menu that will give you the option to create a **new** trigger for that event.
    - Or you can double click an existing trigger to get into an editor on that particular trigger. This will allow you to update the trigger in place as it were, rather than drop and recreate it.