

ADVANCED DATABASE MANAGEMENT SYSTEMS LAB

TRIGGERS

Submitted To,
Lisha Varghese
Assistant Professor
Amal Jyothi College Of Engineering
Kanjirapally

Submitted By,
Teena Rose Mathew
RMCA-B Batch
Roll no: 35

AIM

Create a Trigger for employe table it will update another table salary while updating values

OBJECTIVE

To develop and execute a Trigger for After update/Delete/Insert operations on a table

PROCEDURE

step 1: start

step 2: initialize the trigger.

step 3: On update the trigger has to be executed.

step 4: execute the trigger procedure after updation

step 5: carryout the operation on the table to check for trigger execution.

step 6: stop

PROGRAM

sql>

```
CREATE TABLE `employe` (  
  `emp_id` int(11) NOT NULL,  
  `emp_name` varchar(45) DEFAULT NULL,  
  `dob` date DEFAULT NULL,  
  `address` varchar(45) DEFAULT NULL,  
  `designation` varchar(45) DEFAULT NULL,  
  `mobile_no` int(11) DEFAULT NULL,  
  `dept_no` int(11) DEFAULT NULL,  
  `salary` int(11) DEFAULT NULL,  
  PRIMARY KEY (`emp_id`)  
);
```

Sql>

```
CREATE TABLE `salary` (  
  `employee_id` int(11) NOT NULL,  
  `old_sal` int(11) DEFAULT NULL,  
  `new_sal` int(11) DEFAULT NULL,  
  `rev_date` date DEFAULT NULL,  
  PRIMARY KEY (`employee_id`)  
);
```

sql>

```
CREATE DEFINER=`root`@`localhost` TRIGGER  
`db1`.`personal_updates_AFTER_UPDATE_1`  
AFTER UPDATE ON `employe`  
FOR EACH ROW
```

BEGIN

if(new.salary != old.salary)

then

INSERT INTO salary (employee_id,old_sal,new_sal,rev_date) values

(new.emp_id,old.salary,new.salary,sysdate());

END if;

end;

sql>

update employee set salary=234569 where emp_id=1;

select * from salary;

The screenshot displays the MySQL Workbench interface. The left sidebar shows the 'SCHEMAS' tree with the 'db1' database selected, containing tables like 'employee' and 'salary'. The central SQL editor shows the following queries:

```
update employee set salary=234569 where emp_id=1;
select * from salary;
```

The 'Result Grid' at the bottom shows the output of the 'select * from salary;' query, displaying columns: employee_id, old_sal, new_sal, rev_date. The data rows are:

employee_id	old_sal	new_sal	rev_date
1	10000	10000	2023-08-17
2	10000	10000	2023-08-17
3	10000	10000	2023-08-17

The 'Action Output' pane at the bottom shows the execution log with the following messages:

- 136 17:01:52 Apply changes to employee
- 137 17:02:02 Apply changes to employee
- 138 17:16:52 SELECT * FROM db1.employee LIMIT 0: 1000
- 139 17:17:15 ANALYZE TABLE 'db1'.employee
- 140 17:22:30 update employee set salary=1290 where emp_id=1
- 141 17:22:30 select * from salary LIMIT 0: 1000

The status bar at the bottom indicates the system is running on Windows, with a temperature of 30°C, rain showers, and the time is 05:23 PM on 17-08-2023.

AIM

Create a Trigger for employe table it will update another table personal_updates while updating values

OBJECTIVE

To develop and execute a Trigger for Before and After update/Delete/Insert operations on a table

PROCEDURE

step 1: start

step 2: initialize the trigger.

step 3: On update the trigger has to be executed.

step 4: execute the trigger procedure after updation

step 5: carryout the operation on the table to check for trigger execution.

step 6: stop

PROGRAM

sql>

```
CREATE TABLE `employe` (  
  `emp_id` int(11) NOT NULL,  
  `emp_name` varchar(45) DEFAULT NULL,  
  `dob` date DEFAULT NULL,  
  `address` varchar(45) DEFAULT NULL,  
  `designation` varchar(45) DEFAULT NULL,  
  `mobile_no` int(11) DEFAULT NULL,  
  `dept_no` int(11) DEFAULT NULL,  
  `salary` int(11) DEFAULT NULL,  
  PRIMARY KEY (`emp_id`)  
);
```

Sql>

```
CREATE TABLE `personal_updates` (  
  `emp_id` int(11) NOT NULL,  
  `old_phoneno` int(11) DEFAULT NULL,  
  `new_phoneno` int(11) DEFAULT NULL,  
  `rev_date` date DEFAULT NULL,  
  PRIMARY KEY (`emp_id`)  
);
```

sql>

```
CREATE DEFINER=`root` @`localhost` TRIGGER  
`db1`.`personal_updates_AFTER_UPDATE`  
AFTER UPDATE ON `employe`  
FOR EACH ROW  
BEGIN
```

```
if(new.mobile_no != old.mobile_no)
```

```
then
```

```
INSERT INTO personal_updates (emp_id,old_phoneno,new_phoneno,rev_date)  
values (new.emp_id,new.mobile_no,old.mobile_no,sysdate());
```

```
END if;
```

```
end;
```

```
sql>
```

```
update employee set mobile_no=34566 where emp_id=4 ;
```

```
select * from personal_updates;
```

The screenshot shows the MySQL Workbench interface. The SQL editor contains the following queries:

```
1 * update employee set mobile_no=34566 where emp_id=4 ;  
2 * select * from personal_updates;
```

The Results window displays the output of the second query, showing a table with 4 columns: emp_id, old_phoneno, new_phoneno, and rev_date. The data is as follows:

emp_id	old_phoneno	new_phoneno	rev_date
1	2234568	22345	2021-08-17
4	34566	211475	2021-08-17

The Output window shows the execution log with the following messages:

- 2 23:16:16 Apply changes to employee: No changes detected.
- 3 23:16:24 Apply changes to employee: No changes detected.
- 4 23:18:04 update employee set mobile_no=34566 where emp_name='jocax': Error Code: 1054. Unknown column 'empname' in 'where clause'
- 5 23:18:10 update employee set mobile_no=34566 where emp_name='jocax': Error Code: 1176. You are using safe update mode and you tried to update a table without a WHERE that uses...
- 6 23:19:00 update employee set mobile_no=34566 where emp_id=4: 1 row(s) affected. Rows matched: 1 Changed: 1 Warnings: 0.
- 7 23:19:00 select * from personal_updates LIMIT 0, 1000: 2 row(s) returned.

AIM

Create a Trigger for employee table it will update another table promotions while updating values

OBJECTIVE

To develop and execute a Trigger for Before and After update/Delete/Insert operations on a table

PROCEDURE

step 1: start

step 2: initialize the trigger.

step 3: On update the trigger has to be executed.

step 4: execute the trigger procedure after updation

step 5: carryout the operation on the table to check for trigger execution.

step 6: stop

PROGRAM

sql>

```
CREATE TABLE `employee` (  
  `emp_id` int(11) NOT NULL,  
  `emp_name` varchar(45) DEFAULT NULL,  
  `dob` date DEFAULT NULL,  
  `address` varchar(45) DEFAULT NULL,  
  `designation` varchar(45) DEFAULT NULL,  
  `mobile_no` int(11) DEFAULT NULL,  
  `dept_no` int(11) DEFAULT NULL,  
  `salary` int(11) DEFAULT NULL,  
  PRIMARY KEY (`emp_id`)  
);
```

Sql>

```
CREATE TABLE `personal_updatations` (  
  `emp_id` int(11) NOT NULL,  
  `old_phoneno` int(11) DEFAULT NULL,  
  `new_phoneno` int(11) DEFAULT NULL,  
  `rev_date` date DEFAULT NULL,  
  PRIMARY KEY (`emp_id`)  
);
```

sql>

```
CREATE DEFINER=`root`@`localhost`  
TRIGGER `db1`.`employee_AFTER_UPDATE_1`  
AFTER UPDATE ON `employee`  
FOR EACH ROW  
BEGIN  
  if(new.designation != old.designation)
```

then

```
INSERT INTO promotions (emp_id,old_designation,new_designation,rev_date)
values (new.emp_id,new.designation,old.designation,sysdate());
```

```
END if;
```

```
end;
```

sql>

```
update employee set designation='clk' where emp_id=4 ;
```

```
select * from promotions;
```

The screenshot displays the MySQL Workbench interface. On the left, the 'SCHEMAS' pane shows a list of databases, with 'db1' selected. Under 'db1', several tables are listed, including 'employee', 'personal_updates', 'promotions', 'salary', and 'student'. The 'employee' table is highlighted. The main workspace shows a SQL query editor with the following code:

```
1 * update employee set designation='clk' where emp_id=4 ;
2 * select * from promotions;
```

Below the query editor, the 'Results Grid' displays the output of the 'select * from promotions;' query. The grid has five columns: 'emp_id', 'old_designation', 'new_designation', 'rev_date', and 'rev_date'. The data rows are:

emp_id	old_designation	new_designation	rev_date	rev_date
1	rd	ceo	2021-06-17	
4	ck	ceo	2021-06-17	
1000	1000	1000	1000	

On the right side of the interface, a 'SELECT Syntax' sidebar is visible, providing information about the SELECT statement. At the bottom, the 'Output' pane shows a log of database actions, including updates and selects, with their respective timestamps and durations.