

ADVANCED ADBMS LAB

Ashish Wilson
Roll No:28
S2MCA_A

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

```
CREATE TABLE `db`.`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`));
```

```
CREATE TABLE `db`.`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`));
```

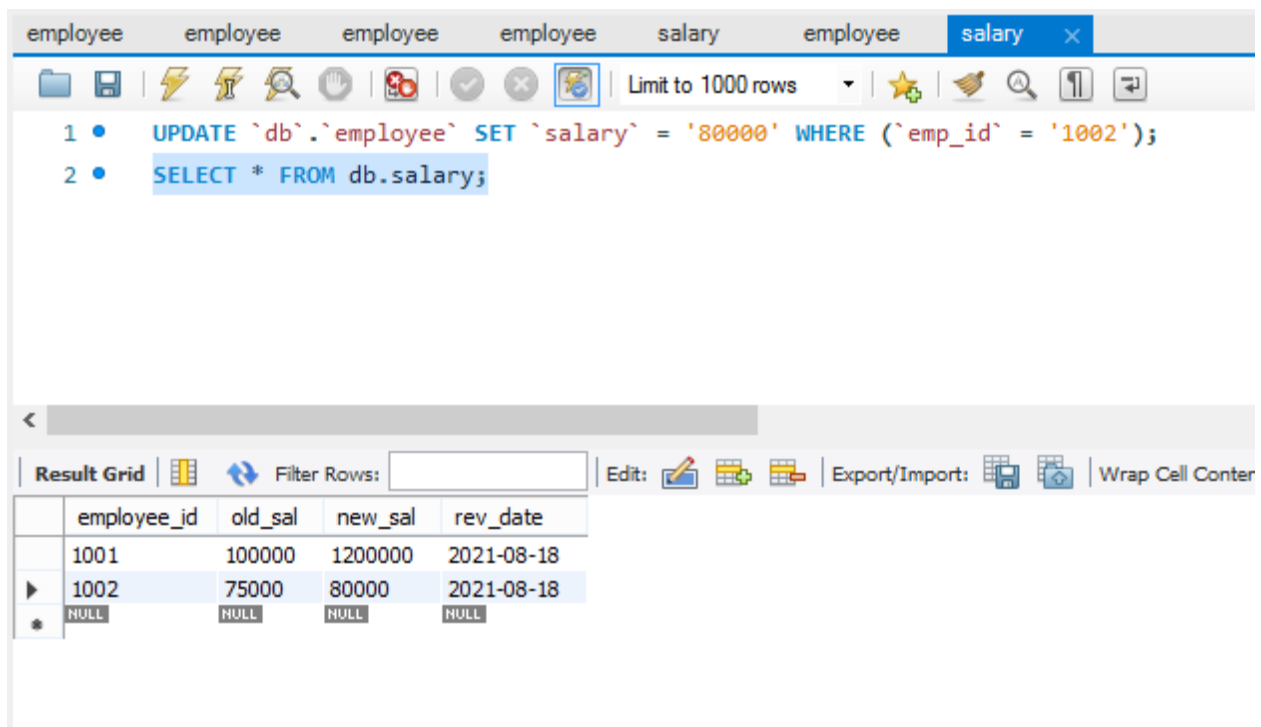
```

CREATE DEFINER=`root`@`localhost` TRIGGER
`db`.`employee_AFTER_UPDATE` AFTER UPDATE ON `employee` 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

UPDATE `db`.`employee` SET `salary` = '80000' WHERE (`emp_id` = '1002');
Select * from db.salary

```



The screenshot shows a database management interface with multiple tabs. The 'salary' tab is active, displaying two SQL queries:

- 1 • `UPDATE `db`.`employee` SET `salary` = '80000' WHERE (`emp_id` = '1002');`
- 2 • `SELECT * FROM db.salary;`

Below the queries, the 'Result Grid' is visible, showing the output of the second query. The grid has four columns: `employee_id`, `old_sal`, `new_sal`, and `rev_date`. The data is as follows:

	employee_id	old_sal	new_sal	rev_date
	1001	100000	1200000	2021-08-18
▶	1002	75000	80000	2021-08-18
*	NULL	NULL	NULL	NULL

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**TABLE**

```
CREATE TABLE `db`.`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`));
```

```
CREATE TABLE `db`.`personal_updatons` (`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`));
```

TRIGGER

```
CREATE DEFINER=`root`@`localhost` TRIGGER  
`db`.`employee_AFTER_UPDATE` AFTER UPDATE ON `employee` FOR  
EACH ROW
```

```
BEGIN
```

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

```
then
```

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

```
END if;
```

```
END
```

```
‘UPDATE `db`.`employee` SET `mobile_no` = '944712130' WHERE (`emp_id` =  
'1005');
```

```
SELECT * FROM db.personal_updatons;
```


AIM

Create a Trigger for employe 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

TABLE

```
CREATE TABLE `db`.`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`) );
```

```
CREATE TABLE `db`.`promotions` ( `emp_id` int(11) NOT NULL,  
`old_designation` varchar(11) DEFAULT NULL, `new_designation` varchar(11)  
DEFAULT NULL, `rev_date` date DEFAULT NULL, PRIMARY KEY  
(`emp_id`));
```

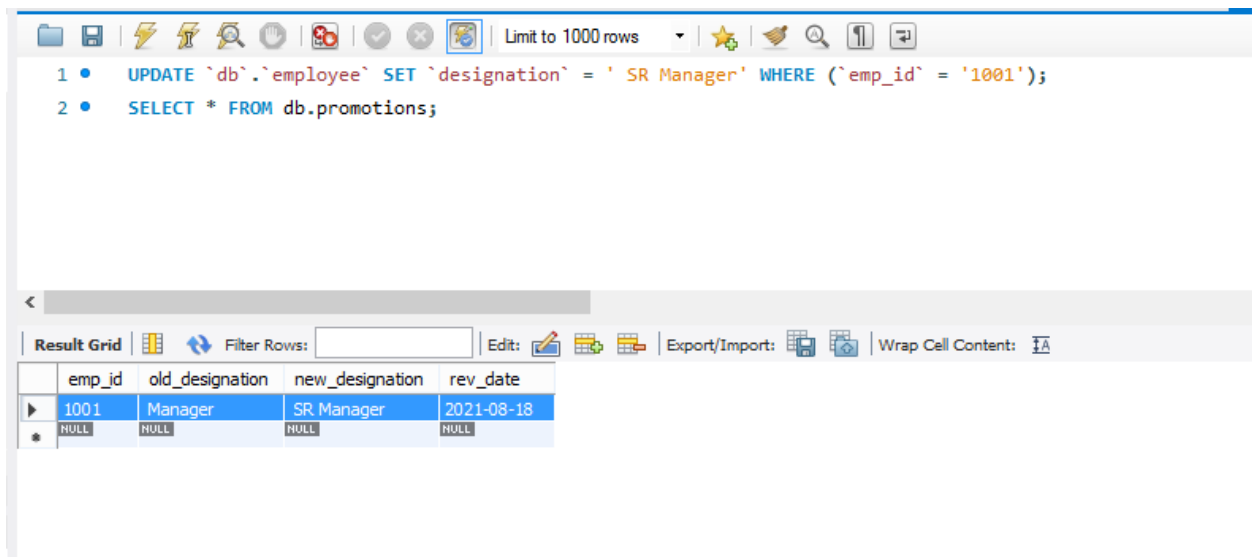
TRIGGER

```
CREATE DEFINER=`root` @`localhost`  
TRIGGER `db`.`employe_AFTER_UPDATE_1`  
AFTER UPDATE ON `employe`  
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;
```

```
UPDATE `db`.`employee` SET `designation` = ' SR Manager' WHERE (`emp_id`
= '1001');
```

```
SELECT * FROM db.promotions;
```



The screenshot shows a database management interface. The top toolbar includes icons for file operations, execution, and search. Below the toolbar, two SQL statements are listed: 1. UPDATE `db`.`employee` SET `designation` = ' SR Manager' WHERE (`emp_id` = '1001'); 2. SELECT * FROM db.promotions; The bottom section displays the 'Result Grid' for the second query. It shows a table with four columns: emp_id, old_designation, new_designation, and rev_date. The first row contains the values 1001, Manager, SR Manager, and 2021-08-18. Below this row, there are two rows with NULL values for all columns, each preceded by an asterisk (*).

emp_id	old_designation	new_designation	rev_date
1001	Manager	SR Manager	2021-08-18
*	NULL	NULL	NULL
*	NULL	NULL	NULL