

## Assignment for SQL Concepts and Fundamentals

1) Write an example for INSTEAD OF TRIGGERS for each category (INSERT, UPDATE, DELETE). Do some meaning actions when it is triggers? Like updating the audit table, setting up the updated time (Do not just display something from database).

### Solution:

#### DATABASE

```
create database portfolio;
use portfolio;
create TABLE stocks( stock_id int, company varchar(20), price int, purchased timestamp);
create TABLE updated_stocks_audit( stock_id int, company varchar(20), old_price int,
new_price int, purchased timestamp, updated timestamp);
create TABLE StocksArchives(stockid int,company varchar(20),price int, purchased timestamp);
```

#### BEFORE INSERT

```
CREATE TRIGGER before_stocks_insert BEFORE INSERT ON stocks FOR EACH ROW SET
@sum = @sum + NEW.price;
```

```
SET @sum = 0;
```

```
INSERT into stocks
```

```
values(123,"Accolite",2324,current_timestamp()),(435,"Quantiphi",1123,current_timestamp()),(5
46,"Workindia",3421,current_timestamp());
```

```
SELECT @sum AS 'Total Stock price';
```

```

1 create database portfolio;
2 use portfolio;
3 create TABLE stocks( stock_id int, company varchar(20), price int, purchased timestamp);
4 create TABLE updated_stocks_audit( stock_id int, company varchar(20), old_price int, new_price int, purchased timestamp, upda
5 create TABLE StocksArchives(stockid int,company varchar(20),price int, purchased timestamp);
6
7
8 CREATE TRIGGER before_stocks_insert BEFORE INSERT ON stocks FOR EACH ROW SET @sum = @sum + NEW.price;
9
10 SET @sum = 0;
11 INSERT into stocks values(123,"Accolite",2324,current_timestamp()),(435,"Quantiphi",1123,current_timestamp()),(546,"Workindi
12 SELECT @sum AS 'Total Stock price';
13
14 delimiter //

```

Result Grid	Filter Rows: Search	Export: [icon]
Total Stock price		
6868		
Result 17		

Action Output	Time	Action	Response
2	19:46:03	create TABLE stocks( stock_id int, company varchar(20), price int, purchased timestamp)	0 row(s) affected
3	19:46:06	create TABLE updated_stocks_audit( stock_id int, company varchar(20), old_price int, new_price int, purchased timestamp,...	0 row(s) affected
4	19:46:09	create TABLE StocksArchives(stockid int,company varchar(20),price int, purchased timestamp)	0 row(s) affected
5	19:46:11	CREATE TRIGGER before_stocks_insert BEFORE INSERT ON stocks FOR EACH ROW SET @sum = @sum + NEW.price	0 row(s) affected
6	19:46:14	SET @sum = 0	0 row(s) affected
7	19:46:16	INSERT into stocks values(123,"Accolite",2324,current_timestamp()),(435,"Quantiphi",1123,current_timestamp()),(546,"Work...	3 row(s) affected Records: 3 Duplicates: 0 War
8	19:46:20	SELECT @sum AS 'Total Stock price' LIMIT 0, 1000	1 row(s) returned

Fig 1: SELECT @sum AS 'Total Stock price';

## BEFORE UPDATE

delimiter //

CREATE TRIGGER before\_stocks\_update BEFORE UPDATE ON stocks  
FOR EACH ROW

BEGIN

IF NEW.price < 0 THEN

SET NEW.price = 0;

ELSEIF NEW.price > 4000 THEN

SET NEW.price = 4000;

END IF;

INSERT INTO updated\_stocks\_audit

values(OLD.stock\_id,OLD.company,OLD.price,NEW.price, OLD.purchased,  
current\_timestamp() );

END; //

delimiter ;

```

UPDATE stocks SET price = 4567 where company="Accolite";
SELECT * from stocks;
SELECT * from updated_stocks_audit;

```

**trigger\***

Limit to 1000 rows

```

17 BEGIN
18 IF NEW.price < 0 THEN
19 SET NEW.price = 0;
20 ELSEIF NEW.price > 4000 THEN
21 SET NEW.price = 4000;
22 END IF;
23 INSERT INTO updated_stocks_audit values(OLD.stock_id,OLD.company,OLD.price,NEW.price, OLD.purchased, current_time
24
25 END;
26 delimiter ;
27
28 UPDATE stocks SET price = 4567 where company="Accolite";
29 SELECT * from stocks;

```

100% 14:29

**Result Grid** Filter Rows: Search Export:

stock_id	company	price	purchased
123	Accolite	4000	2020-01-11 19:46:16
435	Quantiphi	1123	2020-01-11 19:46:16
546	Workindia	3421	2020-01-11 19:46:16

stocks 19

**Action Output**

	Time	Action	Response
7	19:46:16	INSERT into stocks values(123,"Accolite",2324,current_timestamp()),(435,"Quantiphi",1123,current_timestamp()),(546,"Work...	3 row(s) affected Records: 3 Duplicates: 0 War
8	19:46:20	SELECT @sum AS 'Total Stock price' LIMIT 0, 1000	1 row(s) returned
9	19:46:49	SELECT @sum AS 'Total Stock price' LIMIT 0, 1000	1 row(s) returned
10	19:46:54	CREATE TRIGGER before_stocks_update BEFORE UPDATE ON stocks FOR EACH ROW BEGIN IF NEW.price < 0 T...	0 row(s) affected
11	19:46:59	CREATE TRIGGER before_stocks_update BEFORE UPDATE ON stocks FOR EACH ROW BEGIN IF NEW.price < 0 T...	Error Code: 1359. Trigger already exists
12	19:47:04	UPDATE stocks SET price = 4567 where company="Accolite"	1 row(s) affected Rows matched: 1 Changed: 1
13	19:47:07	SELECT * from stocks LIMIT 0, 1000	3 row(s) returned

Fig: SELECT \* from stocks;

trigger\*

```

19      SET NEW.price = 0;
20      ELSEIF NEW.price > 4000 THEN
21          SET NEW.price = 4000;
22      END IF;
23      INSERT INTO updated_stocks_audit values(OLD.stock_id,OLD.company,OLD.price,NEW.price, OLD.purchased, current_time);
24
25  END;
26  delimiter ;
27
28  UPDATE stocks SET price = 4567 where company="Accolite";
29  SELECT * from stocks;
30  SELECT * from updated_stocks_audit;
31

```

100% 14:30

Result Grid Filter Rows: Search Export:

stock_id	company	old_price	new_price	purchased	updated
123	Accolite	2324	4000	2020-01-11 19:46:16	2020-01-11 19:47:04

updated\_stocks\_audit 20

Action Output

	Time	Action	Response
8	19:46:20	SELECT @sum AS 'Total Stock price' LIMIT 0, 1000	1 row(s) returned
9	19:46:49	SELECT @sum AS 'Total Stock price' LIMIT 0, 1000	1 row(s) returned
10	19:46:54	CREATE TRIGGER before_stocks_update BEFORE UPDATE ON stocks FOR EACH ROW BEGIN IF NEW.price < 0 T...	0 row(s) affected
11	19:46:59	CREATE TRIGGER before_stocks_update BEFORE UPDATE ON stocks FOR EACH ROW BEGIN IF NEW.price < 0 T...	Error Code: 1359. Trigger already exists
12	19:47:04	UPDATE stocks SET price = 4567 where company="Accolite"	1 row(s) affected Rows matched: 1 Changed: 1
13	19:47:07	SELECT * from stocks LIMIT 0, 1000	3 row(s) returned
14	19:47:23	SELECT * from updated_stocks_audit LIMIT 0, 1000	1 row(s) returned

Fig: SELECT \* from updated\_stocks\_audit;

## BEFORE DELETE

DELIMITER \$\$

CREATE TRIGGER before\_stocks\_delete

BEFORE DELETE

ON stocks FOR EACH ROW

BEGIN

INSERT INTO StocksArchives(stockid,company,price,purchased)

VALUES(OLD.stock\_id,OLD.company,OLD.price,OLD.purchased);

END\$\$

DELIMITER ;

Delete from stocks where company = "Workindia";

SELECT \* from stocks;

SELECT \* from StocksArchives;

trigger\*

Limit to 1000 rows

```

32 DELIMITER $$
33 CREATE TRIGGER before_stocks_delete
34 BEFORE DELETE
35 ON stocks FOR EACH ROW
36 BEGIN
37     INSERT INTO StocksArchives(stockid,company,price,purchased)
38     VALUES(OLD.stock_id,OLD.company,OLD.price,OLD.purchased);
39 END$$
40 DELIMITER ;
41
42 Delete from stocks where company = "Workindia";
43 SELECT * from stocks;
44 SELECT * from StocksArchives;

```

100% 14:43

Result Grid Filter Rows: Search Export:

stock_id	company	price	purchased
123	Accolite	4000	2020-01-11 19:46:16
435	Quantiphi	1123	2020-01-11 19:46:16

stocks 22

Action Output

	Time	Action	Response
✓ 13	19:47:07	SELECT * from stocks LIMIT 0, 1000	3 row(s) returned
✓ 14	19:47:23	SELECT * from updated_stocks_audit LIMIT 0, 1000	1 row(s) returned
✓ 15	19:47:35	SELECT * from updated_stocks_audit LIMIT 0, 1000	1 row(s) returned
✓ 16	19:47:39	CREATE TRIGGER before_stocks_delete BEFORE DELETE ON stocks FOR EACH ROW BEGIN INSERT INTO StocksArchives(...	0 row(s) affected
✗ 17	19:47:43	CREATE TRIGGER before_stocks_delete BEFORE DELETE ON stocks FOR EACH ROW BEGIN INSERT INTO StocksArchives(...	Error Code: 1359. Trigger already exists
✓ 18	19:47:46	Delete from stocks where company = "Workindia"	1 row(s) affected
✓ 19	19:47:50	SELECT * from stocks LIMIT 0, 1000	2 row(s) returned

Fig: SELECT \* from stocks;

The screenshot shows a SQL IDE with a trigger named 'before\_stocks\_delete' created on the 'stocks' table. The trigger inserts data into 'StocksArchives' before deleting from 'stocks' where the company is 'Workindia'. Below the SQL editor, the 'Result Grid' shows the 'StocksArchives' table with 23 rows. The 'Action Output' pane shows a list of actions and their responses, including a successful query for 'SELECT \* from StocksArchives'.

```

33 CREATE TRIGGER before_stocks_delete
34 BEFORE DELETE
35 ON stocks FOR EACH ROW
36 BEGIN
37     INSERT INTO StocksArchives(stockid,company,price,purchased)
38     VALUES(OLD.stock_id,OLD.company,OLD.price,OLD.purchased);
39 END$$
40 DELIMITER ;
41
42 Delete from stocks where company = "Workindia";
43 SELECT * from stocks;
44 SELECT * from StocksArchives;
45

```

stockid	company	price	purchased
546	Workindia	3421	2020-01-11 19:46:16

Time	Action	Response
15 19:47:35	SELECT * from updated_stocks_audit LIMIT 0, 1000	1 row(s) returned
16 19:47:39	CREATE TRIGGER before_stocks_delete BEFORE DELETE ON stocks FOR EACH ROW BEGIN INSERT INTO StocksArchives(...	0 row(s) affected
17 19:47:43	CREATE TRIGGER before_stocks_delete BEFORE DELETE ON stocks FOR EACH ROW BEGIN INSERT INTO StocksArchives(...	Error Code: 1359. Trigger already exists
18 19:47:46	Delete from stocks where company = "Workindia"	1 row(s) affected
19 19:47:50	SELECT * from stocks LIMIT 0, 1000	2 row(s) returned
20 19:47:59	create database portfolio	Error Code: 1007. Can't create database 'portfolio'
21 19:48:07	SELECT * from StocksArchives LIMIT 0, 1000	1 row(s) returned

Fig: SELECT \* from StocksArchives;

2) Write a stored procedure to paginate the data that comes out of a query

(A pagination is to fetch only limited amount of data from DB according to the user input rather than fetching the entire set of data. User will select the page number and the number of records per page. According to the user input, only a specific set of data is fetched.

Eg: user selects page number as 1 and number of records per page as 10, The database should fetch the first 10 records.

user selects page number as 3 and number of records per page as 10, The database should fetch 10 records starting from 31st record which is the third page)

a) use the input parameter of stored procedure as page number and number of rows per page.

a) A query needs to be executed to display data from Employees Table according to inputs

## **Solution:**

### **assignment.sql**

create database office;

use office;

```
CREATE TABLE Employee (  
    emp_id int primary key,  
    name varchar(20),  
    score decimal(4,2),  
    email varchar(30)  
);
```

```
INSERT into Employee values(13, "Darshan  
Patil",9.45,"darshan.patil@spit.ac.in");
```

```
INSERT into Employee values(32, "Rohit  
Gonsalves",9.65,"rohit.gonsalves@spit.ac.in");
```

```
INSERT into Employee values(45, "Omkar  
Raykar",9.55,"omkar.raykar@spit.ac.in");
```

```
INSERT into Employee values(87, "Rohan  
Pawar",9.69,"rohan.pawar@spit.ac.in");
```

```
INSERT into Employee values(14, "Gaurav  
Yadav",9.62,"darshan.patil@spit.ac.in");
```

```
INSERT into Employee values(21, "Abhigyan  
Nayak",9.85,"abhigyan.nayak@ami.ac.in");
```

```
INSERT into Employee values(7, "Jaspri  
Singh",9.71,"jasprit.singh@nit.ac.in");
```

```
INSERT into Employee values(19, "Rohan  
Jagtap",9.45,"rohan.jagtap@spit.ac.in");
```

```
INSERT into Employee values(26, "Jinay  
Parekh",9.95,"jinay.parekh@spit.ac.in");
```

```
INSERT into Employee values(41, "Lekha  
Sharma",9.91,"lekha.sharma@spit.ac.in");
```

```
INSERT into Employee values(11, "Radnyee  
mhatre",9.55,"radnyee.mhatre@spit.ac.in");
```

```
SELECT * from Employee;
```

```
call pagination(4,2);
```

### **Pagination() procedure**

```
USE `office`;
```

```
DROP procedure IF EXISTS `pagination`;
```

```
DELIMITER $$
```

```
USE `office`$$
```

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `pagination`(IN  
page_no int, IN no_row int)
```

```
BEGIN
```

```
    DECLARE a INT;
```

```
    SET a = (page_no - 1)*no_row;
```

```
    SELECT * from Employee LIMIT a,no_row;
```

```
END$$
```

```
DELIMITER ;
```

### **Results:**



Assignment\*
pagination - Routine
trigger
Limit to 1000 rows

```

8 );
9 INSERT into Employee values(13, "Darshan Patil",9.45,"darshan.patil@spit.ac.in");
10 INSERT into Employee values(32, "Rohit Gonsalves",9.65,"rohit.gonsalves@spit.ac.in");
11 INSERT into Employee values(45, "Omkar Raykar",9.55,"omkar.raykar@spit.ac.in");
12 INSERT into Employee values(87, "Rohan Pawar",9.69,"rohan.pawar@spit.ac.in");
13 INSERT into Employee values(14, "Gaurav Yadav",9.62,"darshan.patil@spit.ac.in");
14 INSERT into Employee values(21, "Abhigyan Nayak",9.85,"abhigyan.nayak@ami.ac.in");
15 INSERT into Employee values(7, "Jasprit Singh",9.71,"jasprit.singh@nit.ac.in");

```

100%
1:1

Result Grid
Filter Rows: Search
Edit:
Export/Import:

emp_id	name	score	email
7	Jasprit Singh	9.71	jasprit.singh@nit.ac.in
11	Radnyee mhatre	9.55	radnyee.mhatre@spit.ac.in
13	Darshan Patil	9.45	darshan.patil@spit.ac.in
14	Gaurav Yadav	9.62	darshan.patil@spit.ac.in
19	Rohan Jagtap	9.45	rohan.jagtap@spit.ac.in
21	Abhigyan Nayak	9.85	abhigyan.nayak@ami.ac.in
26	Jinay Parekh	9.95	jinay.parekh@spit.ac.in
32	Rohit Gonsalves	9.65	rohit.gonsalves@spit.ac.in
41	Lekha Sharma	9.91	lekha.sharma@spit.ac.in
45	Omkar Raykar	9.55	omkar.raykar@spit.ac.in
87	Rohan Pawar	9.69	rohan.pawar@spit.ac.in

Employee 6
Result 7

Action Output

	Time	Action	Response
✓ 14	18:57:50	INSERT into Employee values(11, "Radnyee mhatre",9.55,"radnyee.mhatre@spit.ac.in")	1 row(s) affected
✓ 15	18:57:50	SELECT * from Employee LIMIT 0, 1000	11 row(s) returned
✓ 16	18:57:50	USE `office`	0 row(s) affected
⚠ 17	18:57:50	DROP procedure IF EXISTS `pagination`	0 row(s) affected, 1 warning(s): 1305 PROCEDURE DOES NOT EXIST
✓ 18	18:57:50	USE `office`	0 row(s) affected
✓ 19	18:57:50	CREATE DEFINER='root'@'localhost' PROCEDURE `pagination` (IN page_no int, IN no_row int) BEGIN DECLARE a INT; SE...	0 row(s) affected
✓ 20	18:57:50	call pagination(4,2)	2 row(s) returned

Fig: SELECT \* from Employee;

Assignment\* pagination - Routine trigger

Limit to 1000 rows

```

26 DELIMITER $$
27 USE `office`$$
28 CREATE DEFINER=`root`@`localhost` PROCEDURE `pagination`(IN page_no int, IN no_row int)
29 BEGIN
30     DECLARE a INT;
31     SET a = (page_no - 1)*no_row;
32     SELECT * from Employee LIMIT a,no_row;
33 END$$
34
35 DELIMITER ;
36
37 call pagination(4,2);

```

100% 1:1

Result Grid Filter Rows: Search Export:

emp_id	name	score	email
26	Jinay Parekh	9.95	jinay.parekh@spit.ac.in
32	Rohit Gonsalves	9.65	rohit.gonsalves@spit.ac.in

Employee 6 Result 7

Action Output

	Time	Action	Response
✓ 14	18:57:50	INSERT into Employee values(11, "Radnyee mhatre",9.55,"radnyee.mhatre@spit.ac.in")	1 row(s) affected
✓ 15	18:57:50	SELECT * from Employee LIMIT 0, 1000	11 row(s) returned
✓ 16	18:57:50	USE `office`	0 row(s) affected
⚠ 17	18:57:50	DROP procedure IF EXISTS `pagination`	0 row(s) affected, 1 warning(s): 1305 PROCEDURE DOES NOT EXIST
✓ 18	18:57:50	USE `office`	0 row(s) affected
✓ 19	18:57:50	CREATE DEFINER=`root`@`localhost` PROCEDURE `pagination`(IN page_no int, IN no_row int) BEGIN DECLARE a INT; SE...	0 row(s) affected
✓ 20	18:57:50	call pagination(4,2)	2 row(s) returned

Fig: call pagination(4,2);