Practical 10

Name: Shivam Tawari

Roll no.: A-58

Aim: Exercise 6: Triggers & stored procedures in Database.

Theory:

1. Triggers:

A trigger is a special kind of procedure which executes only when some triggering event such as INSERT, UPDATE, DELETE operations occurs in a table.
Triggers are named database objects that are implicitly fired when a triggering event occurs. The trigger action can be run before or after the triggering event. Triggers are similar to stored procedures but differ in the way that they are invoked.
A Trigger is implicitly invoked whenever any event such as INSERT, DELETE, UPDATE occurs in a TABLE.
Only nesting of triggers can be achieved in a table. We cannot define/call a trigger inside another trigger.
In a database, syntax to define a trigger: CREATE TRIGGER TRIGGER_NAME
Transaction statements such as COMMIT, ROLLBACK, SAVEPOINT are not allowed in triggers.
Triggers are used to maintain referencial integrity by keeping a record of activities performed on the table.
We cannot return values in a trigger. Also, as an input, we cannot pass values as a parameter.

2. Procedures:
A procedure is a combination of SQL statements written to perform a specified tasks. It helps in code re-usability and saves time and lines of code.
A Procedure is explicitly called by user/application using statements or commands such as exec, EXECUTE, or simply procedure_name.
We can define/call procedures inside another procedure.
In a database, syntax to define a procedure: CREATE PROCEDURE PROCEDURE_NAME.
All transaction statements such as COMMIT, ROLLBACK are allowed in procedures.
Procedures are used to perform tasks defined or specified by the users.
We can return 0 to n values. However, we can pass values as parameters.

Program/Queries:
Syntax for creating trigger

```
CREATE [ CONSTRAINT ] TRIGGER name { BEFORE | AFTER | INSTEAD OF } { event [
OR ... ] }
   ON table_name
   [ FROM referenced_table_name ]
   [ NOT DEFERRABLE | [ DEFERRABLE ] { INITIALLY IMMEDIATE | INITIALLY
DEFERRED } ]
```

Name	**Description**
Name	The name of the trigger. A trigger must be distinct from the name of any other trigger for the same table. The name cannot be schema-qualified — the trigger inherits the schema of its table.
Before After Instead Of	Determines whether the function is called before, after, or instead of the event. A constraint trigger can only be specified as AFTER.
Event	One of INSERT, UPDATE, DELETE, or TRUNCATE, that will fire the trigger.
Table_Name	The name of the table or view the trigger is for.
Referenced_Table_Name	The (possibly schema-qualified) name of another table referenced by the constraint. This option is used for foreign-key constraints and is not recommended for general use. This can only be specified for constraint triggers.
Deferrable Not Deferrable Initially Immediate Initially Deferred	The default timing of the trigger.
For Each Row For Each Statement	Specifies whether the trigger procedure should be fired once for every row affected by the trigger event, or just once per SQL statement. If neither is specified, FOR EACH STATEMENT is the default.
Condition	A Boolean expression that determines whether the trigger function will actually be executed.

Function_Name	A user-supplied function that is declared as taking no arguments and returning type trigger, which is executed when the trigger fires.
Arguments	An optional comma-separated list of arguments to be provided to the function when the trigger is executed. The arguments are literal string constants.

[FOR [EACH] { ROW | STATEMENT }]
[WHEN (condition)]
EXECUTE PROCEDURE function_name (arguments)

Parameters:

Syntax of the create function statement:

```
create [or replace] function function_name(param_list)
   returns return_type
   language plpgsql
  as
$$
declare
-- variable declaration
begin
 -- logic
end;
$$
```

```
create function get_film_count(len_from int, len_to int)
returns int
language plpgsql
as
$$
declare
   film_count integer;
begin
   select count(*)
   into film_count
   from film
   where length between len_from and len_to;
   return film_count;
end;
$$;
```

Program/Queries:

```
CREATE OR REPLACE FUNCTION rec_insert1()
RETURNS trigger AS
$$
BEGIN
INSERT INTO StudentFees(Rollno,FirstName,LastName,fees,edittime)
VALUES(NEW.Rollno,NEW.FirstName,NEW.LastName,NEW.fees,current_date);
RETURN NEW;
END;
$$
LANGUAGE 'plpgsql';
CREATE TRIGGER ins_same_rec1
AFTER INSERT
ON Students
FOR EACH ROW
EXECUTE PROCEDURE rec_insert1();
INSERT INTO Students VALUES(89, 'Ajinkya', 'Rahane', '2003-01-03', 4,748392948,
'Delhi',9284);
INSERT INTO Students VALUES(99, 'Virat', 'Kohli', '2002-10-18', 4,9285729168,
'Gurgaon',3827);
select * from StudentFees;
```

Program/Queries:

```
DROP TABLE Students CASCADE;
DROP TABLE studentfees;

CREATE TABLE Students (
 Rollno INT NOT NULL,
 FirstName VARCHAR (100) NOT NULL,
 LastName VARCHAR (100) NOT NULL,
 DateOfBirth DATE NOT NULL,
 Grade INT CHECK(Grade BETWEEN 1 AND 12) NOT NULL,
 ContactDetails BIGINT NOT NULL UNIQUE,
 PRIMARY KEY(Rollno)
);

ALTER TABLE Students ADD City VARCHAR (20);
ALTER TABLE Students ADD fees INT;

INSERT INTO Students VALUES
(28, 'Atharva', 'Khedkar', '2001-10-07', 4, 7987175300, 'Nagpur',10000),
(2, 'Shivam', 'Kumar', '2000-05-15', 4,  9452195859, 'Amravati',15000),
(24, 'Aditya', 'Bobde', '2001-04-12', 4, 8683738291, 'Patna',4999),
(59, 'Krushnakant', 'Bhagwat', '2001-09-18', 4, 7573839281, 'Chandigarh',10000),
(79, 'Aniket', 'Singh', '1999-11-07', 4,9284617472, 'Haryana',3000);
create table StudentFees(Rollno INT NOT NULL,FirstName VARCHAR (100) NOT NULL, LastName VARCHAR (100) NOT NULL,
                         fees integer,edittime date);

CREATE OR REPLACE FUNCTION rec_insert1()
RETURNS trigger AS
$$
BEGIN
INSERT INTO StudentFees(Rollno,FirstName,LastName,fees,edittime)
VALUES(NEW.Rollno,NEW.FirstName,NEW.LastName,NEW.fees,current_date);

RETURN NEW;
END;
$$
LANGUAGE 'plpgsql';

CREATE TRIGGER ins_same_rec1
AFTER INSERT
ON Students
FOR EACH ROW
EXECUTE PROCEDURE rec_insert1();

INSERT INTO Students VALUES(89, 'Ajinkya', 'Rahane', '2003-01-03', 4,748392948, 'Delhi',9284);
INSERT INTO Students VALUES(99, 'Virat', 'Kohli', '2002-10-18', 4,9285729168, 'Gurgaon',3827);
select * from StudentFees;
```

Data Output | Explain | Messages | Notifications

	rollno integer	firstname character varying (100)	lastname character varying (100)	fees integer	edittime date
1	89	Ajinkya	Rahane	9284	2021-10-18
2	99	Virat	Kohli	3827	2021-10-18

Data Output | Explain | Messages | Notifications

	rollno [PK] integer	firstname character varying (100)	lastname character varying (100)	dateofbirth date	grade integer	contactdetails bigint	city character varying (20)	fees integer
1	28	Atharva	Khedkar	2001-10-07	4	7987175300	Nagpur	10000
2	2	Shivam	Kumar	2000-05-15	4	9452195859	Amravati	15000
3	24	Aditya	Bobde	2001-04-12	4	8683738291	Patna	4999
4	59	Krushnakant	Bhagwat	2001-09-18	4	7573839281	Chandigarh	10000
5	79	Aniket	Singh	1999-11-07	4	9284617472	Haryana	3000
6	89	Ajinkya	Rahane	2003-01-03	4	748392948	Delhi	9284
7	99	Virat	Kohli	2002-10-18	4	9285729168	Gurgaon	3827

Conclusion: Hence, we have learnt and performed the creation of trigger along with function.