WEEK-6

① Aim:- To Execute SQL Server stored procedure in SQL Developer?.

Procedure:-

Theory:- A stored procedure is a set of (T-SQL) statements needed in times when we are having the repetitive usage of the same query. when there is a need to use a large query multiple times we can create a stored procedure once and execute the same whenever needed instead of writing the whole query again.

Syntax:- For creating a stored procedure

```
CREATE PROCEDURE (or CREATE PROC) proc_name
AS
Begin
    END QUERY
END
```

Step1:- we creating a Database. For this use the below command to create a database named GeeksforGeeks.

Query:-

```
CREATE DATABASE GeeksforGeeks:
```

Output:-

Step 2:- To use GeeksforGeeks database use the below command

Query:- USE GeeksforGeeks

Output:-

Step 3:- Now we creating a table. Create a table Student_details with 3 columns using the following SQL Query:

Query:- CREATE TABLE Student_details (
stu_id VARCHAR (8),
stu_name VARCHAR (20),
stu_cgpa DECIMAL (4,2));

Output:-

Step4:- The query for inserting rows into the table. Inserting rows into Student_details table using the following SQL Query.

Query:-
INSERT INTO Student_details VALUES ('40001', 'PRADEEP', 9.6), ('40002', 'ASHOK', 8.2), ('40003', 'PAVAN KUMAR', 7.6), ('40004', 'NIKHIL', 8.2), ('40005', 'RAHUL', 7.0);

Output:-

Step5:- Viewing the inserted data
Query:- Select * FROM Student_details;
Output:-

* Query to create a stored procedure to view the table :
Query:-
CREATE PROCEDURE view_details
AS

```
BEGIN
   Select * From Student_details;
END
```

Output:-

For executing a stored procedure we use the below syntax:

Syntax:-
```
Exec PROC_name
or
Execute PROC_name
or
PROC_name
```

Query:- EXECUTE view_details

Output:-

- Query to create a stored procedure that takes the argument as stu_id and displays the cgpa of that id.

Query:-
```
CREATE PROCEDURE get_student_cg-details
@stu_id VARCHAR(20)
AS
BEGIN
   SELECT stu_id, stu_cgpa FROM student_details
    WHERE stu_id = @stu_id
END
```

Output:-

Query:-

```
EXECUTE  get_student_cg_details  '40002'
```

Output:-

Result.-

② Aim:- To create Procedures in PL/SQL.

Procedure:-

*PL/SQL is a block-structured language that enables developers to combine the power of SQL with procedural statements. A stored procedure in PL/SQL is nothing but a series of declarative SQL statements which can be stored in the database catalogue. A procedure can be thought of as a function or a method. They can be invoked through triggers, other procedures, or applications on Java, PHP etc.
All the statements of a block are passed to Oracle engine all at once which increases processing speed and decreases the traffic.

Advantages:-

- They result in performance improvement of the application. If a procedure is being called frequently in an application in a single connection, then the compiled version of the procedure is delivered.
- They reduce the traffic between the database and the application, since the lengthy statements are already fed into the database and need not be sent again and again via the application.
- They add to code reusability, similar to how functions and methods work in other languages such as C/C++ and Java.

Disadvantages:-

- Stored procedures can cause a lot of memory usage. The database administrator should decide an upper bound as to how many stored

are feasible for a particular application.

- MySQL does not provide the functionality of debugging the stored procedures.

Syntax to create a stored procedure:-

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- comments --
CREATE PROCEDURE procedure_name
=,
=,
=
AS
BEGIN
--Query--
END
GO
```

Program:-

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE GetStudentDetails
    @StudentID int =0
AS
BEGIN
    SET NOCOUNT ON;
    SELECT FirstName, LastName, BirthDate, City, Country,
    From Students Where Student ID = @StudentID
END
GO
```

Output:-

Syntax to modify an existing stored procedure.

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- Comments --
ALTER PROCEDURE Procedure_name
  =,
  =,
  =
AS
BEGIN
-- Query --
END
GO
```

Program:-

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE GetStudentDetails
      @StudentID int=0
AS
BEGIN
      SET NOCOUNT ON;
      SELECT FirstName, LastName, [illegible], City, [illegible]
      FROM Students where StudentID = @StudentID
END
GO
```

Output:-

Syntax to drop a Procedure:-

```
DROP    PROCEDURE    procedure_name
```

Program:-

```
DROP    PROCEDURE    GetStudentDetails
```

Output:-

Result:- Creating procedures in PL/SQL has been completed successfully.

(3)

Aim:- To create Triggers in SQL Server.

Procedure:-

What is a Trigger:-

A Trigger is a special kind of stored procedure that executes in response to certain action on the table like insertion, deletion or updation of data. It is a database object which is bound to a table and is executed automatically. You can't explicitly invoke triggers. The only way to do this is by performing the required action on the table that they are assigned to.

Types of Triggers:-

There are three action query types that you use in SQL which are INSERT, UPDATE and DELETE. So, there are three types of triggers and hybrids that come from mixing and matching the events and timings that fire them. Basically triggers are classified in to two main types:-

(i) After Triggers (For Triggers)
(ii) Instead of Triggers

(i) After Triggers:-

These triggers run after an insert, update or delete on a table. They are not supported for views. After Triggers can be classified further into three types as:

a. AFTER INSERT Trigger
b. AFTER UPDATE Trigger
c. AFTER DELETE Trigger.

Let's create After triggers. First of all, let's create a table and insert some sample data. Then, on this table, I will be attacking several triggers.

Program:-

```
CREATE TABLE Employee-Test
(
Emp-ID INT Identity,
Emp-name Varchar (100),
Emp-Sal Decimal (10,2)
)
INSERT INTO Employee-Test Values ('Anees', 1000),
('Rick', 1200), ('John', 1101, ('Stephen', 1300),
('Maria', 1400);
```

Output:-

I will be creating an AFTER INSERT Trigger which will insert the rows inserted into the table into another audit table. The main purpose of this audit table is to record the changes in the main table. This can be thought of as a generic audit trigger.

Now, create the audit table as:-

```
CREATE TABLE
Employee-Test-Audit
(
Emp-ID int,
Emp-name varchar (100),
Emp-Sal decimal (10,2),
Audit-Action varchar (100),
Audit-TimeStamp datetime
)
```

Output :-

(a) After Insert Trigger :-

This trigger is fired after an Insert on the table.

Let's create the trigger as:

Program :-

```
CREATE TRIGGER trgAfterInsert on [dbo].[Employee_Test]
FOR INSERT
AS
    declare @empid int;
    declare @empname varchar(100);
    declare @empsal decimal (10, 2);
    declare @audit_action varchar(100);
    Select @empid = i.Emp_ID from inserted ;;
    Select @empname = i.Emp_Name from inserted ;;
    Select @empsal = i.Emp_Sal from inserted ;;
    Set @audit_action = 'Inserted Record -- After Insert
                         Trigger. ';
    Insert INTO Employee_Test_Audit (Emp_ID, Emp_Name,
    Emp_Sal, Audit_Action, Audit_Timestamp) values
    (@empid, @empname, @empsal, @audit_action,
     getdate ());
    print ('After Insert trigger fired.')
GO
```

Output :-

The CREATE TRIGGER statement is used to create the trigger. The ON clause specifies the table name on which the trigger is to be attached. The FOR INSERT specifies that this is an AFTER INSERT trigger. In place of FOR INSERT, AFTER INSERT can be used. Both of them mean the same.

In the trigger body, table named inserted has been used. This table is a logical table and contains the row that has been inserted. I have selected the fields from the logical inserted table from the row that has been inserted in to different variables and finally inserted those values into the Audit table.

To see the newly created trigger in action. Let's insert a row in to the main table as:

Program:-

```
INSERT INTO Employee_Test values ('Chris', 1500);
```

Now, a record has been inserted into the Employee_Test table. The AFTER INSERT trigger attached to this table has inserted the record into the Employee_Test_Audit as:

Output:-

(b) AFTER UPDATE Trigger:-

This trigger is fired after an update on the table. Let's create the trigger as:

Program:-

```
Create Trigger trg AfterUpdate ON [dbo].[Employee_Test]
FOR UPDATE
AS
    declare @empid int;
    declare @empname varchar(100);
    declare @empsal decimal(10,2);
    declare @audit_action varchar(100);
    Select @empid = i.Emp_ID from inserted i;
    Select @empname = i.Emp_name from inserted i;
    Select @empsal = i.Emp_Sal from inserted i;
    if Update(Emp_Name)
      Set @audit_action = 'Updated Record -- After Update
      Trigger.';
    If Update(Emp_Sal)
      Set @audit_action = 'Updated Record -- After
      Update Trigger.';
    INSERT INTO Employee_Test_Audit (Emp_ID, Emp_name,
    Emp_Sal, Audit_Action, Audit_Timestamp) VALUES(
    @empid, @empname, @empsal, @audit_action,
    getdate());
    Print 'After Update Trigger Fired.'
    Go
```

Output:-

The AFTER UPDATE Trigger is created in which the updated record is inserted into the audit table. There is no logical table inserted/updated like the logical table inserted,

We can obtain the updated value of a field from the Update (column_name) function. In our trigger we have used, if update (Emp Name) to check if the column Emp Name has been updated. We have similarly checked the column Emp_Sal for an update.

Let's update a record column and see what happens.

Program:-

```
Update Employee_Test Set Emp_Sal = 1550 where
Emp_ID = 6
```

This inserts the row into the audit table as:-

Output:-

(c) After Delete Trigger:-

This trigger is fired after a delete on the table.

Let's create the trigger as:-

Program:-

```
Create Trigger trgAfterDelete ON [dbo].[Employee_Test]
After Delete
AS
    declare @empid int;
    declare @empname varchar(100);
    declare @empsal decimal(10,2);
    declare @audit_action varchar(100);
    Select @empid = d.Emp_ID from deleted d;
    Select @empname = d.Emp_Name from deleted d;
    Select @empsal = d.Emp_Sal from deleted d;
    set @audit_action = 'Deleted -- After Delete
    Trigger.';
```

```
Insert into Employee_Test_Audit (Emp_ID, Emp_Name,
Emp_Sal, Audit_Action, Audit_Timestamp) values (
@empid, @empname, @empsal, @audit_action,
getdate ());
Print 'After delete Trigger fired.'
GO.
```

In this trigger, the deleted record's data is picked from the logical deleted table and inserted into the audit table. Let's fire a delete on the main table. A record has been inserted into the audit table as:

OUTPUT:-

All the triggers can be enabled / disabled on the table using the statement

```
ALTER TABLE Employee_Test {Enable / Disable} Trigger All
```

Specific Triggers can be enabled or disabled as:

```
ALTER TABLE Employee_Test
DISABLE TRIGGER
trgAfterDelete
```

OUTPUT:-

This disables the After Delete Trigger named trgAfterDelete on the specified table -

(ii) Instead Of Triggers:-

These can be used as an interception for anything that anyone tried to do on our table or view. If you define an Instead Of Trigger on a table for the delete operation, they try to delete rows, and they will not actually get deleted.

INSTEAD OF TRIGGERS can be classified further into three types as:

a. INSTEAD OF INSERT Trigger
b. INSTEAD OF UPDATE Trigger
c. INSTEAD OF DELETE Trigger.

Let's create an Instead of Delete Trigger:-

Program:-

```
Create Trigger tgg InsteadOfDelete ON [dbo].[Employee_Test]
INSTEAD OF DELETE
AS
   declare @emp_id int;
   declare @emp_name varchar(100);
   declare @emp_Sal int;
   Select @emp_id = d.Emp_ID from deleted d;
   Select @emp_name =d.Emp_Name from deleted d;
   Select @emp_Sal =d.Emp_Sal from deleted d;
   BEGIN
      if (@emp_Sal > 1200)
      begin
         RAISERROR ('Cannot delete where
         Salary > 1200', 16, 1);
         ROLLBACK;
```

```
    end
    else
    begin
       delete from Employee_Test where Emp_ID = @emp_id;
       COMMIT;
  INSERT INTO Employee_Test_Audit (Emp_ID, Emp_Name,
  Emp_Sal, Audit_Action, Audit_Timestamp) VALUES (
  @emp_id, @emp_name, @emp_sal,
  'Deleted -- Instead of Delete Trigger.', getdate());
  Print 'Record Deleted -- Instead of Delete Trigger.'
      end
   END
  GO
```

OUTPUT:-

This trigger will prevent the deletion of records from the table where Emp_Sal > 1200. If such a record is deleted, the Instead of Trigger will rollback the transaction, otherwise the transaction will be committed. Now, let's try to delete a record with the Emp_Sal > 1200 as

Program:-

```
delete from Employee_Test where Emp_ID = 4
```

This will print an error message as defined in the RAISE ERROR statement as:-

Output:-

And this record will not be deleted.

Result:- Creating Triggers in SQL has been executed successfully.