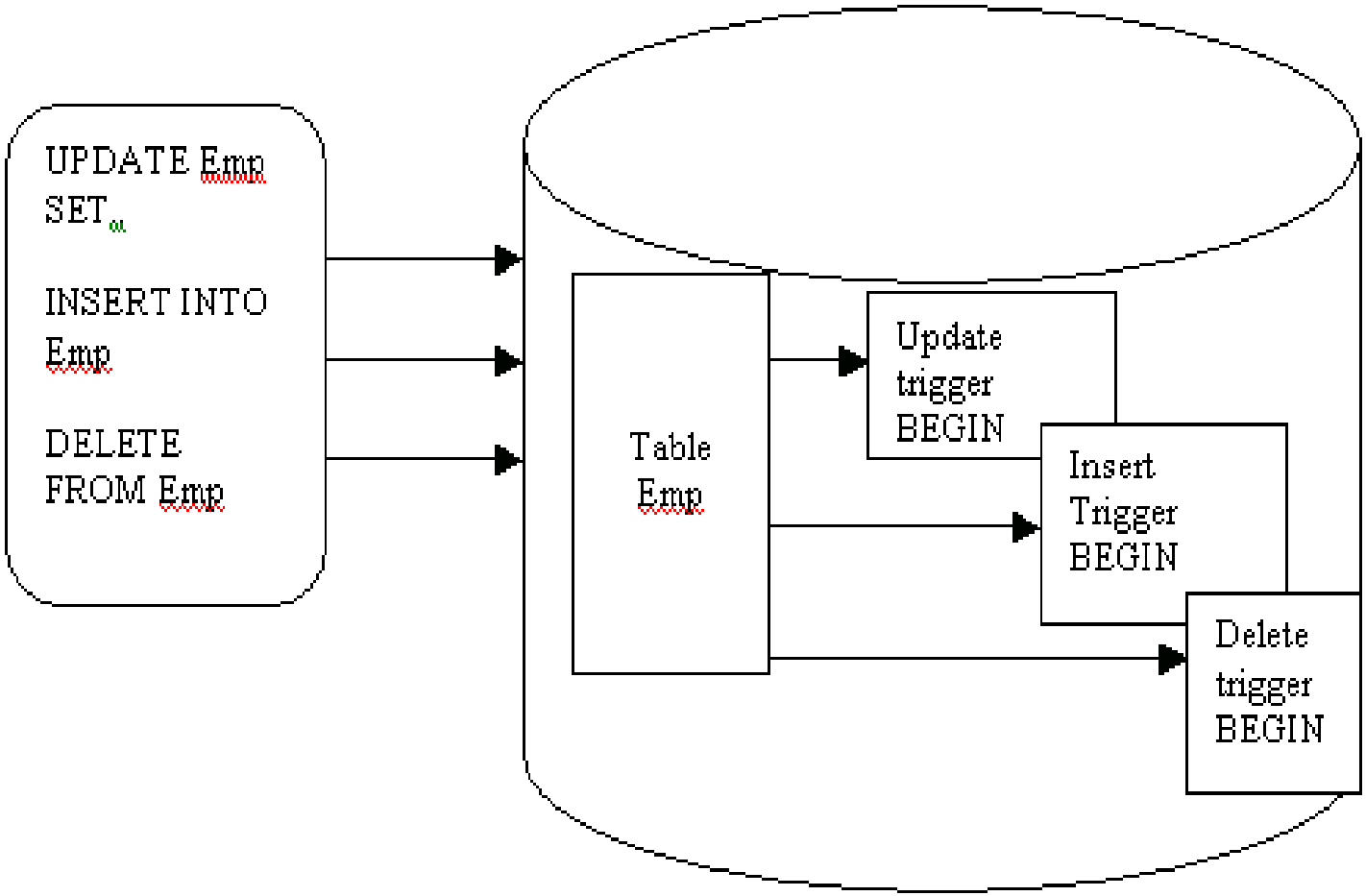
Triggers

By
Parteek Bhatia
Assistant Professor
Dept of Comp Sc & Engg
Thapar University
Patiala

Introduction to Triggers

- A database trigger is a stored procedure that is fired when an INSERT, UPDATE, or DELETE statements is issued against the associate table. The name trigger is appropriate, as these are triggered (fired) whenever the above-mentioned commands are executed. A trigger defines an action the database should take when some database related event occurs..
- A trigger is automatically executed without any action required by the user. A stored procedure on other hand needs to be explicitly invoked. This is the main difference between a trigger and a stored procedure.

Working of Trigger



Use of Database Triggers

- To derive column values automatically.
- To enforce complex integrity constraints.
- To enforce complex business rules.
- To customize complex security authorizations.
- To maintain replicate tables.
- To audit data modifications.

Database Trigger V/s Procedure

Database Triggers	Procedures
Triggers do not accept parameters	Procedures can accept parameters
A trigger is automatically executed without any action required by the user.	A stored procedure on other hand needs to be explicitly invoked.

Parts of a Trigger

- Triggering event or Statement.
- Trigger constraint (Optinal)
- Trigger action

Types of Triggers

Oracle has the following types of triggers depending on the different applications:

- Row Level Triggers
- Statement Level Triggers
- Before Triggers
- After Triggers

Row Level Triggers

- Row level triggers execute once for each row in a transaction. The commands of row level triggers are executed on all rows that are affected by the command that enables the trigger.
- For example, if an UPDATE statement updates multiple rows of a table, a row trigger is fired once for each row affected by the UPDATE statement. If the triggering statement affects no rows, the trigger is not executed at all. Row level triggers are created using the FOR EACH ROW clause in the CREATE TRIGGER command.

Application of row level trigger

- Consider a case where our requirement is to prevent updation of empno 100 record cannot be updated, then whenever UPDATE statement update records, there must be PL/SQL block that will be fired automatically by UPDATE statement to check that it must not be 100, so we have to use Row level Triggers for that type of applications.

Statement Level Triggers

- Statement level triggers are triggered only once for each transaction. For example when an UPDATE command update 15 rows, the commands contained in the trigger are executed only once, and not with every processed row. Statement level trigger are the default types of trigger created via the CREATE TRIGGER command.

Application:

- Consider a case where our requirement is to prevent the DELETE operation during Sunday. For this whenever DELETE statement deletes records, there must be PL/SQL block that will be fired only once by DELETE statement to check that day must not be Sunday by referencing system date, so we have to use Statement level Trigger for which fires only once for above application.

Before and After Trigger

- When a trigger is defined, you can specify whether the trigger must occur before or after the triggering event i.e. INSERT, UPDATE, or DELETE commands.
- BEFORE trigger execute the trigger action before the triggering statement. BEFORE triggers are used when the trigger action should determine whether or not the triggering statement should be allowed to complete. By using a BEFORE trigger, you can eliminate unnecessary processing of the triggering statement.
- For example: To prevent deletion on Sunday, for this we have to use Statement level before trigger on DELETE statement.
- AFTER trigger executes the trigger action after the triggering statement is executed. AFTER triggers are used when you want the triggering statement to complete before executing the trigger action.
- For example: To perform cascade delete operation, it means that user delete the record fro one table, but the corresponding records in other tables are delete automatically by a trigger which fired after the execution of delete statement issued by the user.

Trigger types

There are 12 types.

- BEFORE INSERT row
- BEFORE INSERT statement
- AFTER INSERT row
- AFTER INSERT statement
- BEFORE UPDATE row
- BEFORE UPDATE statement
- AFTER UPDATE Row
- AFTER UPDATE statement
- BEFORE DELETE row
- BEFORE DELETE statement
- AFTER DELETE row
- AFTER DELETE statement

Syntax

```
CREATE [OR REPLACE] TRIGGER trigger_name
{BEFORE | AFTER}
{| DELETE | [OR] INSERT | [OR] UPDATE [OF column_name,...]}
ON table_name
[REFERENCING {OLD AS old, NEW AS new}]
[FOR EACH ROW [WHEN condition]
DECLARE
Variable declaration;
Constant declaration;
BEGIN
        PL/SQL subprogram body;
[EXCEPTION
        exception PL/SQL block;]
END;
```

Triggering event

Triggering restriction

Triggering action

Create a trigger for the emp table, which makes the entry in ENAME column in uppercase.

```
CREATE OR REPLACE TRIGGER upper_trigger
BEFORE INSERT OR UPDATE OF ename ON emp
FOR EACH ROW
BEGIN
   :new.ename : = UPPER (:new.ename);
END;
```

Create a trigger on the emp table, which shows the old values and new value of ename after any updation on ename of emp table.

```
CREATE OR REPLACE TRIGGER EMP_UPDATE
AFTER UPDATE OF ENAME ON EMP FOR EACH ROW
BEGIN
    DBMS_OUTPUT.PUT_LINE('OLD NAME: ' ||:OLD.ENAME);
    DBMS_OUTPUT.PUT_LINE('NEW NAME:' ||:NEW.ENAME);
END;
```

Create a trigger to implement the primary key constraint on column eno of table emptemp (eno, ename, job, sal, deptno).

```
CREATE OR REPLACE TRIGGER PRIMARY_KEY
BEFORE INSERT ON EMPTEMP
FOR EACH ROW
DECLARE
    E EMPTEMP.ENO%TYPE;
BEGIN
    IF (:NEW.ENO IS NULL) THEN
    RAISE_APPLICATION_ERROR(-20002, 'PRIMARY KEY VIOLATION
    BECAUSE IT CANNOT BE NULL');
END IF;
    SELECT ENO INTO E FROM EMPTEMP WHERE ENO=:NEW.ENO;
    RAISE_APPLICATION_ERROR(-20003, 'PRIMARY KEY VIOLATION
    BECAUSE IT SHOULD BE UNIQUE');
EXCEPTION
    WHEN NO_DATA_FOUND THEN
    NULL;
END;
```

Error Handling in Triggers

Syntax:

RAISE_APPLICATION_ERROR (*error_number, message);*

Here:

error_number

- It is a negative integer in the range-20000 to –20999

message

- It is a character string up to 2048 bytes in length

This procedure terminates procedure execution, rolls back any effects of the procedure, and returns a user-specified error number and message.

Create a trigger to implement the foreign key constraint on column deptno of table emptemp (empno, ename, job, sal, deptno) which refer to the deptno column of table depttemp (deptno, dname, loc).

```
CREATE OR REPLACE TRIGGER FOREIGN_KEY
BEFORE INSERT ON EMPTEMP
FOR EACH ROW
DECLARE
    DNO DEPTTEMP.DEPTNO %TYPE;
BEGIN
    SELECT DEPTNO INTO DNO FROM DEPTTEMP WHERE
    DEPTNO=:NEW.DEPTNO;
    NULL;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
    RAISE_APPLICATION_ERROR(-20004, 'FOREIGN KEY VIOLATED
    BECAUSE VALUE IS NOT FOUND IN THE PARENT TABLE');
END;
```

Create a trigger on the emp table, which store the empno and operation in table auditor for each operation i.e. Insert, Update and Delete.

```
CREATE OR REPLACE TRIGGER EMP_AUDIT
AFTER INSERT OR UPDATE OR DELETE ON EMP
FOR EACH ROW
 BEGIN
     IF INSERTING THEN
          INSERT INTO AUDITOR VALUES(:NEW.EMPNO,'INSERT');
     ELSIF UPDATING THEN
          INSERT INTO AUDITOR VALUES(:NEW.EMPNO,'UPDATE');
      ELSIF DELETING THEN
           INSERT INTO AUDITOR VALUES(:OLD.EMPNO,'DELETE');
     END IF;
 END;
```

Note: The same functionality can be achieved with the BEFORE type trigger also.

Create a trigger so that no operation can be performed on emp table on Sunday.

```
CREATE OR REPLACE TRIGGER EMP_SUNDAY
BEFORE INSERT OR UPDATE OR DELETE ON EMP
BEGIN
IF RTRIM(UPPER(TO_CHAR(SYSDATE,'DAY')))='SUNDAY' THEN
    RAISE_APPLICATION_ERROR(-20022,'NO OPERARTION CAN BE
    PERFORMED ON SUNDAY');
END IF;
END;
```

Create a trigger, which verify that no record has the commission value greater than, salary of an employee in table emp.

```
CREATE OR REPLACE TRIGGER REC_CHECK
BEFORE INSERT OR UPDATE ON EMP FOR EACH ROW
BEGIN
    IF :NEW.SAL<:NEW.COMM THEN
    RAISE_APPLICATION_ERROR(-20000,'RECORD IS ILLEGAL');
END IF;
END;
```

Create a trigger, which verify that updated salary of employee must be greater than his/her previous salary.

```
CREATE OR REPLACE TRIGGER UPDATE_CHECK
BEFORE UPDATE ON EMP
FOR EACH ROW
BEGIN
IF :NEW.SAL<:OLD.SAL THEN
    RAISE_APPLICATION_ERROR(-20000,'NEW SALARY CANNOT
    BE LESS THAN OLD SALARY');
END IF;
END;
```

Enabling/Disabling/Dropping a Trigger

ALTER TRIGGER <trigger name> DISABLE / ENAMBLE

Dropping Triggers

DROP TRIGGER trigger_name;

Example

DROP TRIGGER emp_update;

Thanks

Lets Implement it in Lab Session