ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 10, October 2016

Trigger for Monitoring, Tracking & Database Security – From Theory to Practice

Falguni Parsana

Asst. Professor, Dept. of Computer Science & I.T, Shree M &N VIRANI SCIENCE COLLEGE, Atmiya Group of Institutions, Kalawad Road, RAJKOT, GUJARAT, India

ABSTRACT: This paper will enhance triggers techniques improving the effectiveness of database security and transactional Security challenges. The aim of paper is to define the following guidelines when designing triggers: Trigger for centralized, global operations that should be fired for the triggering statement, regardless of which user issues the statement and when. Trigger guaranteed that when a specific operation is performed, related actions are performed. Trigger for Database Security that prevent users to logon after business hours. Trigger for Transactional Security process prevent users to perform invalid transaction into the table.

KEYWORDS: Database Security, Trigger, Transactional Security, Audit trailing

I. INTRODUCTION

Database Security is most essential part for sensitive data. We can prevent unauthorized users for entering invalid data. We can prevent other users for logon process after working hours. There are different levels of securities which I elaborate here,

1) Database Security:
2) Transactional Security:

- *In Database Security,* we can prevent user to logging in an unauthenticated way.

- *Transactional Security* works on certain actions like insert update and delete. It can prevent other unauthorized users to perform any activity into the database. It monitor and gather data about specific database activities for example, the database administrator can gather statistics, about in which time the data is modified, in which time the data is deleted, which transaction is performed by whom and when.

II. LITERATURE SURVEY

Yaya Itai [1] suggested an approach which explores effective code of Database Security. It comes under two processes, one is to prevent logon security and other is to prevent table access after logon. So Transactional Security is also an essential part in any organization or companies. So here I included Transactional security that is implemented as restrictions on who can INSERT, UPDATE, DELETE or SELECT from a table. It allows you to think of security entirely in terms of table access. Using triggers plus table security gives the tightest possible integration of security and business logic, and puts your entire security framework on a simple and consistent basis.

III. WHAT IS DATABASE TRIGGER?

A Database Trigger is a procedure based on Events i.e., Insert, Update and Delete. It automatically executed or fires whenever any events occur in the table. So when certain action takes place, trigger fires automatically. Database triggers can be associated with a table, schema or database.
Triggers will be able to record logon activities of specified users and their attempts to make changes to database objects in an audit trail to a database table or external table. These are the unique business requirements.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

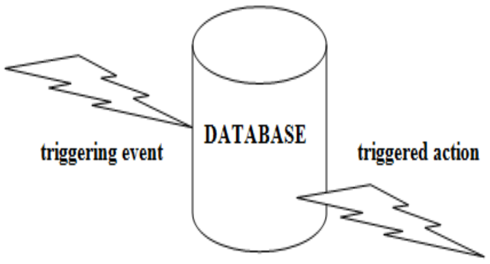
International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 10, October 2016

The trigger is also used for maintaining the integrity of the information on the database. For example, when a new row (representing a new employee) is added to the employees table, that new row should also be created in the tables of the taxes, vacations and salaries.
Before we get to the first example, we need to review the four basic options that are available when you define a trigger. These four options will be used in the examples below, and this quick review can serve as a reference when you review the examples.

- INSERT, UPDATE and DELETE Triggers can be defined for each of these operations, and most databases allow you to assign define a trigger to fire on combinations, like INSERT and UPDATE but not DELETE. Most servers also allow you to define multiple triggers to fire on each event [1].
- BEFORE or AFTER. A trigger can be defined to fire before or after the operation. Triggers that fire before an operation can modify the data being written or reject the operation. Triggers that fire afterwards are good for writing to history tables for backup purpose.
- ROW or STATEMENT TRIGGER. Sometimes Triggering Events affect multiple rows. So it becomes necessary that trigger fires on each affected row. So for maintaining backup, we use Row level trigger. Statement level trigger does not fire on every affected row. So generally statement level trigger is used to give messages on screen after some event occurs.
- SECURITY. It lets you define security just once in terms of table access. No other user can perform any activity without permission.



Example 1:

The code below comes from my class where we discuss several types of triggers. This trigger is something I've implemented over the years for students and is the centre point of my triggers chapter in my Advanced PL/SQL course. This is an auditing trigger. Here the goal is to track who inserted a row and who last updated or deleted the row and when. The idea here is to see who might have entered erroneous data.

Let's start with the 4 maintenance fields:

```
Create table emp_backup_tracking
(
        empno   varchar2(10),
        name            varchar2(20),
        action_dt       date,
        action_nm       varchar2(10),
        username        varchar2(20)  );
```

These four fields will be used to track who inserted the row and when and who last updated/Deleted the row and when.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

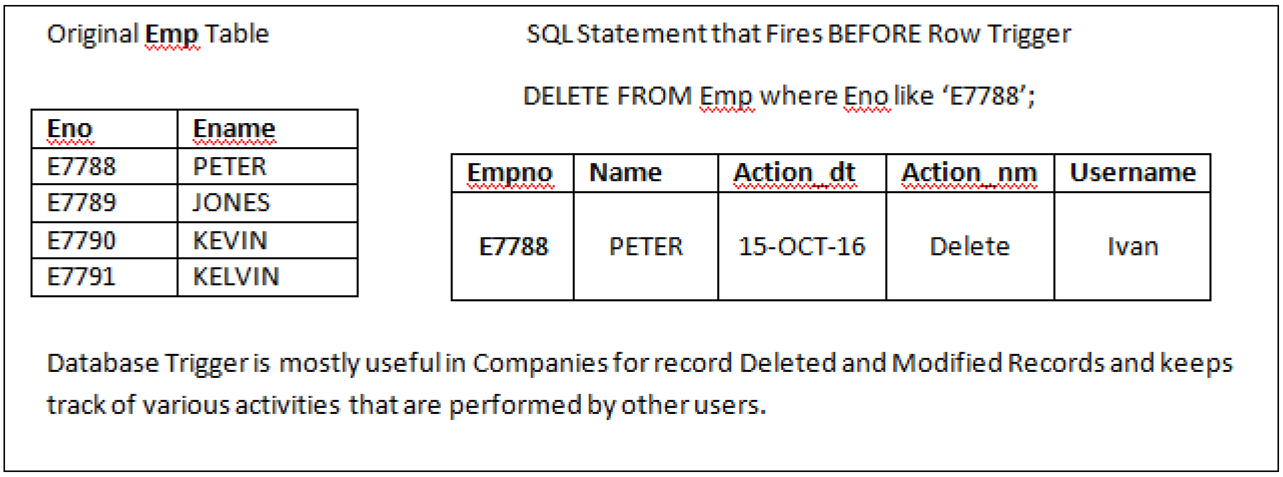
(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 10, October 2016

```
create or replace trigger t_backup_emp before insert or update or delete on emp for each row
declare
        operation varchar2(8);
begin
        if updating then
                operation:='update';
        elsif deleting then
                operation:='delete';
        elsif inserting then
                operation:='insert';
        end if;
insert into emp_backup values(:old.eno,:old.ename,sysdate,operation,user);
end;
/
```

This trigger simply modifies the contents of these four fields before the row is posted to the database. The code executes quickly because basically it is just one IF statement.
Notice the ‘before insert or update or delete’ clause, this is the DML operation that will execute this code. The ‘for each row’ trigger perform on each row affected by an update or delete event.
Triggers can see the data before (: old) the data was changed and the new data being posted (: new).
Notice the INSERTING logic is setting the INSERTED fields to the current user (pseudo column USER is the current logged in user) and SYSDATE, and setting the UPDATED fields to NULL. Since this is a ‘before’ trigger, this code essentially overwrites any attempt to insert any other user or time stamp. Likewise with the UPDATING logic…the original INSERTED fields are reset to their original values and the UPDATED fields are maintained with the current logged in user and time stamp. Below is the process of my work:

Tracking Process:



Original Emp Table

Eno	Ename
E7788	PETER
E7789	JONES
E7790	KEVIN
E7791	KELVIN

SQL Statement that Fires BEFORE Row Trigger

DELETE FROM Emp where Eno like ‘E7788’;

Empno	Name	Action_dt	Action_nm	Username
E7788	PETER	15-OCT-16	Delete	Ivan

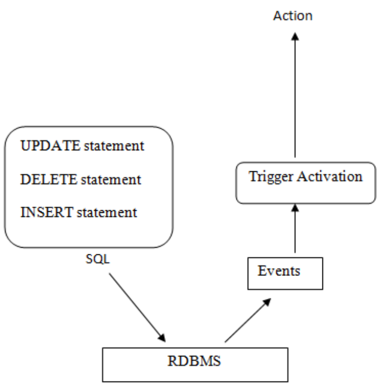
Database Trigger is mostly useful in Companies for record Deleted and Modified Records and keeps track of various activities that are performed by other users.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 10, October 2016



Triggering Flow

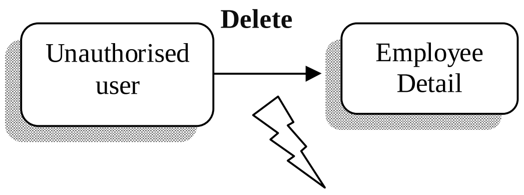
Example 2:
Following example is based on security to avoid invalid transactions by other unauthorized or unauthenticated users.

The below trigger code is to prevent Delete operation on emp table.

```
create or replace trigger mytrigger before
delete on emp
begin
raise_application_error(-20000, 'sorry we
cannot delete any record from this table');
end;
```

Explanation

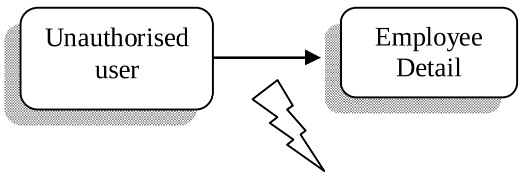
- In the above example raise_application_error is a predefined customized function of exception handling which contains 2 parameters that is error number and user-defined error message.
- Error number is the range between -40000 to -20999
- Parameter represent error message which can give maximum 1048 character.



Cannot delete any record from emp table

The below trigger code is to prevent Update operation on emp table.

```
create or replace trigger mytrig before update on emp
begin
raise_application_error(-20000, 'sorry we cannot perform any DML operation on this     table');
end;
```



Cannot update any record from emp table

IV. DATABASE SECURITY

Integrity, Confidentiality and availability are the hallmarks of database security [1]. It concern with:

Who should have the right to access data in non working hours?
Who should have the right to access data in week end?
What portion of all the data should a particular user be able to access?
What operations should an authorized user be able to perform on the data in week end?
What portion of all the data should a particular user be able to access?
What operations should an authorized user be able to perform on the data?

Example 3:

The following trigger code helps to prevent logon by authorized users. It prevents DML operations on a table after regular business hours.

This below Logon trigger allows only connects from the IP-addresses 192.168.2.122, 192.168.2.123, and 192.168.2.233.

```
            /* LOGON TRIGGER */

CREATE OR REPLACE TRIGGER logon_trigger AFTER LOGON ON DATABASE

BEGIN

IF SYS_CONTEXT('USERENV','IP_ADDRESS') not in ('192.168.2.122','192.168.2.123','192.168.2.233') THEN
            RAISE_APPLICATION_ERROR(-20003,'You are not allowed to connect to the database');
END IF;

IF (to_number(to_char(sysdate,'HH24'))< 8) and (to_number(to_char(sysdate,'HH24')) >18) THEN
            RAISE_APPLICATION_ERROR(-20005,'Logon not allowed after business hours.');
END IF;

/* insert data into an audit-table so table must be created first !!!*/

INSERT INTO user_log VALUES(user, sys_context('USERENV','SESSIONID'), sys_context('USERENV','HOST'),null, null, null,
sysdate, to_char(sysdate, 'hh24:mi:ss'), null, null, null);

END;
/
```

REFERENCES

1. Yaya Itai, "Trigger and Database Security", International Journal of Computers & Technology www.cirworld.com Volume 4 No. 1, Jan-Feb, 2013 ISSN 2277-061

BIOGRAPHY

Parsana Falguni is an Assistant Professor in the Computer Science & Information Technology Department, at Atmiya College of Information Technology & Science, Saurashtra University, Rajkot (Gujarat). She received Master of Computer Application (MCA) degree in 2006. Her research interests are Relational Database Management System, Database Programming. She distributed many tutorials and case studies to the students for practice aspects. Also she had published one book to provide reference & knowledge of Relational Database Management System to students.

Her current research topic is 'Database Trigger' that covers the Transactional Security system and Database Security System in the real world.