THE ESSENTIALITIES OF A TRIGGER: AN INTEGRAL PART OF A DATABASE SECURITY

ABDULRAHMAN YUSUF
Department of Computer Science, Faculty of Science,
Yobe State University Damaturu, Yobe State, Nigeria
Email: audura33@yahoo.com **Phone:** +234-806-611-3431

Abstract
Enforcing security in a database is a matter of great concern as the number of data are accumulated, stored and shared electronically. Internal threat on sensitive data in an organization could leads to the loss of confidentialities and integrity of data. A database trigger which is procedural code that is automatically executed in response to certain events on a particular table, view, schema, or the database itself is an aspect of database security. This paper aimed to investigate and uncover the essentialities, applicability and level of security measures provide by database triggers using SQL/PLSQL programming in Oracle 11g where a simple but complex relational database of a certain concertina scenario were developed. The applied trigger techniques in the created database applications have shown that triggers could check some security measures on the database, among others are collection of statistics for database access, auditing activities of database users, restriction of DML operations on table at unwanted period of time, prevention of invalid transaction and enforcement of different complex security authorization. In general, the applied trigger techniques tested have seemed to prove many benefits of triggers in dealing with internal threats in the database.

Keywords: Database, Trigger, Security, Oracle 11g, Integrity and Schema

Introduction

Data or information which is usually kept in a database are very crucial assets to any organization for analysis and strategic decision making and this could determine the future of an organization. Sheik (2017) mentioned that ensuring information security in DBMS is of great concern as databases are the basic to many business and government associations. Therefore, secrecy, integrity and reliability of data need to be maintained using any possible means of security termed Database Security. Database Security is concerned with the used of collective measures of information security control to protect databases (potentially including the data, the database applications or stored functions, the database systems, the database servers and the associated network link) against compromises of the confidentiality, integrity and availability (Yang, 2009). According to Anon (2011a) database security is a process through which the confidentiality, integrity and availability of the database can be safeguarded. Unauthorized access or entry to a database server means a loss of confidentiality; unauthorized modification to the available data means a loss of integrity and lack of access to the database services means a loss of availability. Security of database is being hampered by affecting any of these basic facets. Sensitive information and data are the bedrock of any organization therefore worth proper security measures and maintenance.

According to Mazer (2006) "…*database used to store sensitive information are now the target of numerous regulations requiring accountability for how data is handled. To meet this challenge, organizations should implement strong database auditing practice.*" Database auditing is considered to be very important security practice following authentication and authorization procedure. Mazer further defines "*Database auditing is the ability to continuously monitor, record analyze and report on all user-level database activities- so as*

to be able to answer questions 'who did what to which data. When and how?". In most organization data integrity and confidentiality are being violated either intentionally or otherwise (Neorem, Hammed & Usman, 2011). In practicing Database auditing, Database triggers are mostly used.

The magnitude of database security requirement varies from one organization to another depending on the data and information sensitivity being maintained. Various technologies and tools are available for enforcing security measures to different facets of database. However, no particular tool can guarantee total protection towards every sort of hazards and harms (Anon, 2011b).

Statement of Problem

Application of security should be both application and database centric (Carl, 2012) for checking illegal access and internal threats respectively. However, more often, application developers do enforce security only at frontend that is application centric where user uses user name and password for access permission, forgotten to maximize security on the actual databases. This leads to compromise data/information security such as confidentiality, integrity and availability especially by insiders. With application of database security especially triggers, database administrators could handle the issues of organizational internal threat where only those who are allowed to do particular action to which data, when and how could do so; above all it could allow the monitoring as well as restricting the activities of application users or database users.

Significance of the study

This research work may have a number of significance which include:

(i) The study would generally disclose the important of triggers at glance for security enforcement on the databases that hold the essential information of an organization.
(ii) The work could serve as a guide to both application developers and students of computer on how to technically and properly apply database triggers on various database application for securing databases.
(iii) The study will also show that application of security should both application and database centric for maintaining confidentiality, integrity and availability.
(iv) The study will show how careful allocation of role-base to the database users could help in checking security in the database.

Review of Related Works

Role-based Access Control (RBAC)

Permission to access or not to access certain resources in the database is very crucial in information management system. Systematic permission allocated to users on business operational system does provide security that protect information and data from both unauthorized users as well as legitimate but illegal access users. RBAC appears to be most recent mechanism use to ensure that only those users with authorized permissions have access to database resources (Qing & Bin, 2010; Cheng, 2012). RBAC has *users*, *roles* and *protected object* as its three important parts (Cheng, 2012). Although RBAC serves as an alternative and solution of self-access control or Discretionary Access Control (DAC) and Mandatory Access Control (MAC) but also has its own defect as opposes by Cheng (2012) the "role" based permission in RBAC, gives too much role of power to a user that should have minimal power (least privilege user). Therefore, more convenient granular access control permission is required to do away with this practical deficiency, which is fine-grained access control.

Database Trigger

Trigger is just like a stored procedure, trigger is stored in the database and it can be invoked recurrently. As opposed to store procedure, database administrator can enabled or disabled trigger. It is automatically invoked by database when enabled but it does not fire when disabled. Trigger only fires whenever its triggering event occurs (Oracle, 2020a). A database trigger is a procedural code that is automatically executed in response to certain events on a particular table, view, schema, or the database itself as can be visualised in Figure 1. Hence, the integrity of the data/information on the database can be maintained using trigger. For example, erroneous data entry, sabotage, and other malicious data manipulations can fire the trigger program (Itai, *et al.,* 2013). With the help of triggers, an administrator can maintain crucial database auditing tasks such as: **a**) reporting any updates that make change to a sensitive column value, and **b**) sustaining a history of changes to a sensitive column (for instance salary). Triggers are declaratively specified in a query independent manner to perform an action when specific data items are accessed (Fabbri, *et al*, 2000). Database Triggers and database auditing seemed to be inseparable security techniques as the former is the sub-function of the latter, as it has been observed that in procedural codes for auditing, triggers are mostly embedded.

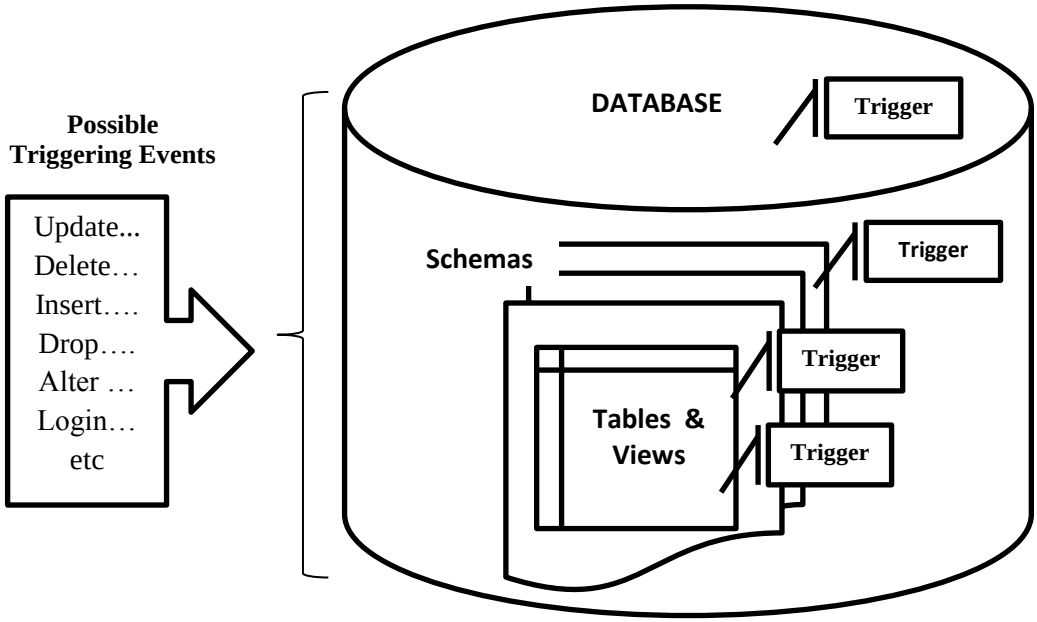

Figure 1: Conceptual application of triggers in a database

Triggers are important especially when you want to record customized information which is before-and-after changes to a table or view. One important thing about trigger is that you do not need to have auditing enabled for it to work, nor does it matter what type of auditing you do have enabled. The trigger performs well outside of the database audit functionality. Hence, the following are guidelines if you want to create audit triggers:

(i) **Avoid creating trigger to write data to the SYS.AUD$ table.** This is to avoid affecting standard auditing as you try to write values to SYS.AUD$ and the trigger does not work as expected or you modify the SYS.AUD$ table contents. The SYS.AUD$ table is an Oracle Database-owned table, and only Oracle Database should write to it.

(ii) **Try as possible, to create an AFTER trigger.** This is to avoid record keeping or audit processing unnecessarily. If triggering statement are not successfully executed or if no records are found, then the AFTER trigger will never be fired.

(iii) **Create the trigger as either an AFTER row or AFTER statement trigger**. Which to use, AFTER row or AFTER statement triggers depends on the information to

be audited. For instance, row triggers deliver value-based auditing for each table row. Triggers need to contain a reason code for issuing the audited SQL statement, which can be worthwhile in both row and statement-level auditing situations.

The Need for Using a Trigger

Triggers appear to be integral part of database security. In real world practice, once a need to have a very good and secured system database arises, triggers can have a very great quarter to contribute. Triggers do play a very crucial role in checking and responding to particular changes that occur to database systems (Lee and *et al*, 2003). System security implementations are promoted with the help of implemented auditing mechanism on the database system (Noreem, Hammed and Usman, 2011). They further explain there is a hierarchy for putting security in place which is *authentication, authorization, encryption and auditing based on triggers***.** Carl (2013) illustrated authentication/identification, authorization and auditing as the main facets of security where: i) Authentication/Identification - determines who are you; ii) Authorization – examines what you can see or do and iii) Auditing – if you are authenticated and authorized *what, when, who, where and how* the activities are performed and monitored. Most auditing actions are integrated with triggers. Carl further stated that triggers can handle more complex business rules which include security authorisations, comprehensive integrity checking, prevention of invalid transactions, auditing mechanisms, transparent event logging and maintenance of derived data values stored in the database.

Triggers guarantee for avoiding or alerting for an action, when a specific operation is performed or related actions are performed. Trigger allows for customization of your database management system. For instance, you can apply triggers for the following action: auto generation of virtual column values; log event; collect statistical data on a table access; amend table data for DML - Data Manipulation Language statements which are issued against views; allowing for enforce referential integrity when child and parent tables are on different nodes of a distributed database; alerting and stopping invalid transaction; alerting and averting DML operations on table after working hours; and allowing for enforce complex business or referential integrity rules that you cannot define with constraints. According to Oracle (2011) Triggers may not be reliable security mechanisms, because they are programmatic and easy to disable. For higher security assurance, it is better to use Oracle database vault. However, disabling the trigger is by database user with appropriate authorization, therefore can be handled by careful allocation of authorization.

Trigger Types

Different DBMS provides different types of triggers but they seem to be similar. Although some DBMS provide very rich security measures while others provide very limited. The research focuses on Oracle DBMS which is used for practical application of this research work. According to Oracle (2011), there are four types which include a DML trigger, a System trigger, a Conditional trigger and an INSTEAD OF triggers**.** DML Trigger is any trigger created on table or view but its triggering event is composed using DLM statements (Delete, Insert, and Update). *System Trigger* can be created on either a schema or a database and the triggering event is composed using either Data Definition Language (DDL) statements (Create, Alter, and Drop) or database operation statements (Servererror, Startup, Logoff, Logon, and Shutdown). A *Conditional Trigger* has an SQL condition that is specified by a When-clause that the database does evaluate for every row being affected by the triggering statement. An INSTEAD OF Trigger can be either a DML trigger created on a view or a system trigger defined on a *Create* statement; and the *Instead Of* trigger is fired by the database instead of running the triggering statement. Oracle (2020a) divides

conditional trigger to be simple if it has one timing point or compound if it is with more than one timing points.

Carl (2013) said triggers has two types: triggers based on tables and triggers based on views. Triggers based on tables could be either *before* (statement or for each row) or *after* (statement or for each row). Triggers based on views are INSTEAD OF triggers. However, Carl has not mentioned system triggers.

Structures of a Trigger and its Creation

Trigger is usually created with having four parts: Trigger type; Triggering Event which could be DML or DDL statement; Trigger Restriction which is the condition determines firing (applicable to ROW level triggers) and Trigger Action (body written in PL/SQL). Trigger can be created using CREATE TRIGGER statement. One can specify the *triggering event* based on *triggering statements* and the element on which they act. The trigger is *created on* or *defined on* the item, which could be either a table, a view, a schema, or the database. A *timing point* can also be specified, to determine when the trigger fires, *before* or *after* the triggering statement runs and whether it fires for each row that affects by the triggering statement. By default, a trigger is in enabled state once it is created. Furthermore, at the same timing point at least two triggers can be created in Oracle DB but their order of execution are indeterminable (Hall, 2000). Below is the structural format.

```
CREATE [OR REPLACE] TRIGGER trigername
    {BEFORE | AFTER}
    {DELETE | INSERT | UPDATE [OF columns]}
        [OR {DELETE | INSERT |UPDATE [OF columns]}]...
ON table
   [FOR EACH ROW [WHEN condition]]
   [REFERENCING [OLD AS old] [NEW AS new]]
PL/SQL block
```

Materials and Methods

This has been approached qualitatively using theoretical and practical methods.

Theoretical Method

The research used academic and research literatures from various sources among others are journals, books, e-journals, e-books, reports, Oracle releases documentations, and conference proceedings. The information and data sources were mostly from Summon library catalogue, and other popular and reputable databases, such as science direct, ACM digital library, Google scholar, Safari digital library, and so on. The study further used the found existing facts for practical application as explained below.

It has been observed that there are various types of trigger techniques provides by Oracle DBMS. The studies exposed the essentialities of triggers and trigger based auditing appropriate for securing database records as in the case of this research work.

Experimental Method

Investigations have been carried out on how Oracle 11g DBMS can be used for application of database triggers on a developed database of a certain scenario. The studies analyzed and created the relational schema for the business rules of the concertina scenario in Oracle DBMS where different triggers types and techniques were used to investigate the contribution of database trigger and the level of security that it could provide to the database system. Therefore, the following tools are used for practicality.

Design

For any system to be developed there is a need to have clear view of its specifications which should be analyzed for designing the required system to meet the specifications. The same were applied for this research work where an appropriate relational schema for the business specifications of the scenario were designed after clear understanding and analysis. A well designed relational database that could help in putting everything into reality for the application of appropriate triggers. Application of appropriate trigger actions solely depend on the application of policies in business rules on the system (or database Objects).

Carl (2013) gives scenario as assuming a company called Concertina in UK is organizing and running repeatedly a scheduled concert involving a single principal artist (or group) in large venues around the country. Every concert has a name, a duration of at least 2 but not more than 5 hours, a type of either classical, rock, or pop and a cost that varies between £30 and £500. The concert as an event can be repeatedly run at different venue. For instance, the CHILI concert could be run a number of times at various venues. Each of the run concert is called an event for that particular concert. The date of every event and its venue are recorded. A Customer can book onto the event but Concertina must record the customer's name, gender, telephone number, and address. Any time a customer attends an event, his arrival time is recorded, and a customer with a car can park at the official car park but the vehicle registration number is recorded too. Any Customer should rank the concert (from 0 to 5) as an evaluation at any attended event and Concertina should record the evaluation from every customer in attendance. It is not necessary that all events have customers registered on them, and a customer can only attend the concert of his will. Each event is held at an approved venue that has a name, a maximum capacity of at most 60,000 seats and a postcode.

Software

The database was designed and created based on the scenario using SQL and PL/SQL languages in Oracle 11g database server express edition. According to Oracle (2020b) Oracle express edition is totally free to download, easy-of-use with allot of features for a database administrator, a developer, an educator or anyone concern with database.. SQL and PL/SQL Plus languages as a foundation to all Oracle application development were used in creating the relational database tables as well as populating them with test data. The policies applied on the system were also created using SQL and PL/SQL languages. The entity relationship model of the scenario is shown in Figure 2.

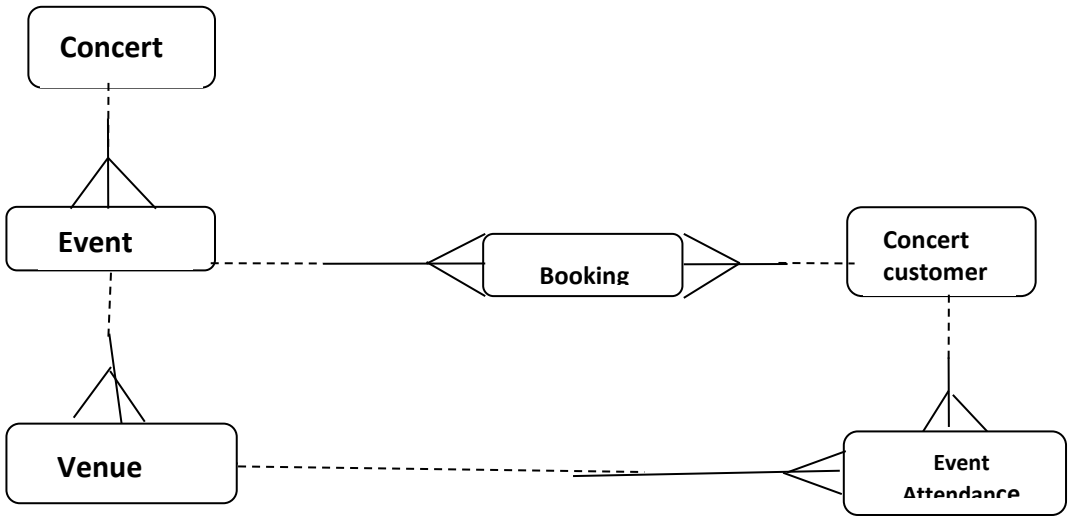

Figure 2: Concertina Scenario ER Model

SQL and PL/SQL languages.

It has been cleared from Oracle website that the keys to all Oracle application development are SQL and PL/SQL languages. This is due to the roles play for provisions of interaction with Oracle database server. Each of SQL and PL/SQL languages has its own playing roles in that interaction. While PL/SQL language allows for triggers creation and call, procedures and functions storage as well as invoking PL/SQL package, but SQL is used for modification and querying Oracle databases. The command line tool for editing SQL and PL/SQL commands as well as formatting querying results and options setting is SQL*Plus.

Trigger Creation

Having understood that security should not be application based rather it should be database-centric. "Database security should be database centric not application centric" (Carl, 2012). The application of security to the database for this research work focused on the action of triggers. Generally, the working principle of the system could be seen in *figure 3* below.

Some triggers would be created on the database system to enforce security policy for better understanding of important of triggers in securing the database. An INSTEAD OF trigger, A conditional trigger, System trigger and DML triggers are different types of trigger as explained in section 2.4. The trigger-based could be created to test for the following:

1. Apply a trigger to allow only registered users of the DB system to manipulate his role-based information.
2. Apply a trigger to alert a user, DBA excluded for any prohibited action.
3. Apply a trigger to disallow any action at particular time such as during weekend by a customer.
4. Apply a trigger to disallow deletion of a record by a user (not DBA) after certain period of time.
5. Apply a trigger to disallow dropping of any object on a particular schema by a user, DBA excluded
6. Apply a trigger to remind the customer certain DML statement he/she is trying to execute.

Implementation and Testing

To actualize and present what seem reliable from the identified roles or applications of trigger techniques, the database application of the scenario was created and experimented as explained below. These experiments were aimed to verify trigger-based action and its level of security provision in the database. The ER diagram was formed as shown in Figure 3. The tables and their associated constraints were created as well as the test data that were used to populate the tables.

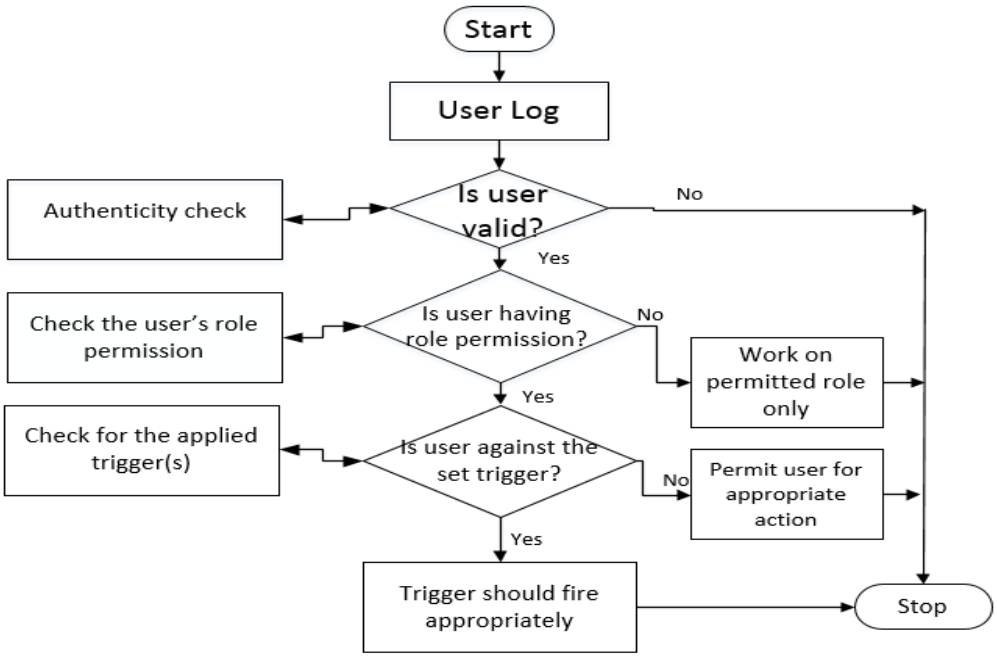

Figure 3: Flowchart of the System for the Trigger Actions

Creation of application users and roles allocations

For easy tracking of who did what to which data, when and how? Careful creation of database users and roles allocations could help greatly. For practical application of this research some users were created and roles were also carefully allocated. As revealed in the literature, the RBAC provides better management of database users. Therefore, this section tries to put RBAC into practice where some roles and privileges have been granted.

Implementation of Triggers

The identified types of triggers and their applications were implemented. The implemented triggers and trigger-based auditing on the DB system is evaluated to ensure what have been mentioned in section 3.4.

DML trigger

Example 1: Create a trigger to ensure that only a customer who has booked for a concert can attend that concert event.

```
CREATE OR REPLACE TRIGGER not_in_booking_list
BEFORE INSERT OR UPDATE ON evt_attendance
FOR EACH ROW
DECLARE
        booking_list    VARCHAR2(16);
BEGIN
        SELECT custr_id INTO booking_list
                FROM evt_attendance
                WHERE custr_id = :NEW.custr_id;
EXCEPTION
        WHEN no_data_found THEN
                RAISE_APPLICATION_ERROR(-20010, 'Invalid Customer: He has not  booked
                for the concert');
END;
```

Eg1-Test1: INSERT INTO OPS$1223793.evt_attendance(attend_id, arrival_time, vehicle_reg_number, evaluation, evt_id, custr_id)
VALUES(402, 13:45, 'FAN 16', 4, 8, 305);

1 row created

Eg1-Test2: INSERT INTO OPS$1223793.evt_attendance
VALUES(402, 13:45, 'KAN 16', 4, 8, 309);

The created trigger is tested with both correct and wrong data as shown above, in **Eg1-Test1** the customer id 305 which is already exist in the booking table as the security check is done. This has shown that the customer has already booked the concert. But **Eg1-Test2** resulted in an error 'Invalid Customer: He has not booked for the concert' because after the security check by the trigger; it has found that the customer id 309 does not exist in booking table which shows that customer has not booked for the concert.

Example 2: Create trigger that when a customer gives zero evaluation the details of his attendance (customer name, attended concert, date of the event, venue and evaluation) will be placed in the audit table.

Below is the trigger after audit table concert_audit was created.

```
CREATE OR REPLACE TRIGGER conct_audit1
        AFTER INSERT OR UPDATE OF evaluation ON evt_attendance
        FOR EACH ROW
    DECLARE
            custr_name_audt             VARCHAR2(14);
            conct_name_audt             VARCHAR2(14);
            evt_date_audt       DATE;
            ven_name_audt               VARCHAR2(14);
            evaluation_audt             NUMBER(2);
        BEGIN
            SELECT custr_name, conct_name, evt_date, ven_name,
                    evaluation INTO custr_name_audt, conct_name_audt,
                    evt_date_audt, ven_name_audt, evaluation_audt
                    FROM concert c, event e, venue v,
                    evt_attendance a,conct_customer cus
            WHERE c.conct_id = e.conct_id
                    AND e.evt_id = v.evt_id
                    AND v.ven_id = a.ven_id
                    AND a.custr_id = cus.custr_id;
            IF :NEW.evaluation = 0 THEN
                    INSERT INTO concert_audit
                            VALUES (custr_name_audt, conct_name_audt,
                                    evt_date_audt, ven_name_audt ,
                                    evaluation_audt);
            END IF;
        END;
```

Eg2-Test1: INSERT INTO evt_attendance VALUES (434, '12:57', 'FAN 16', 0, 10, 305);

The created trigger has been tested. Assuming the customer who attended the event concert and gave zero evaluation, in this case some details associated with that customer from various tables of the database have been captured and put into the audit table.

Example 3: Trigger for alerting and stopping an action which is not allowed in the database. For instance, ensure that no concert can be run within the month of August:

```
CREATE OR REPLACE TRIGGER conct_august_not
BEFORE INSERT OR UPDATE OF evt_date ON event
FOR EACH ROW
BEGIN
        IF (TO_CHAR(:NEW.evt_date, 'MONTH') LIKE 'AUG%') THEN
                RAISE_APPLICATION_ERROR(-20111, 'NO  EVENT OF CONCERT
                                                IS ALLOWED TO RUN IN AUGUST');
        END IF;
END;
```

Eg3-Test1: INSERT INTO event VALUES (987, '15 - AUG - 2013', 125);

Result: ERROR at line 1:

Eg3-Test2: INSERT INTO event VALUES (987, '15 - JAN - 2013', 125);

Result: 1 row created.

In **Eg3-Test1** as expected, the trigger does not allow an event concert to be organized in the month august and pop up 'NO EVENT OF CONCERT IS ALLOWED TO RUN IN AUGUST'. So, with the exception of August an event can be organized in any month as tested in **Eg3-Test2**.

Example 4: The trigger in created to enforce security by raising exceptions whenever user tries to book concert during weekends.

```
CREATE OR REPLACE TRIGGER concert_booking_permit
  BEFORE INSERT OR UPDATE ON booking
  DECLARE
   Booking_not_on_weekends EXCEPTION;
   PRAGMA EXCEPTION_INIT (Booking_not_on_weekends, -3003);

       --check to ensure not weekends
   IF (TO_CHAR(Sysdate, 'DAY') = 'SUN' OR
      TO_CHAR(Sysdate, 'DAY') = 'SAT') THEN
        RAISE Booking_not_on_weekends;
   END IF;

 EXCEPTION
   WHEN Booking_not_on_weekends THEN
   Raise_application_error( -22405, 'Booking of a concert is'
         || 'only done during working days');
END;
```

Eg4-Test1: INSERT INTO booking VALUES (987, Sysdate, 125)

To show the effectiveness of this trigger, it was tested with both correct and wrong entry. Testing was done during working days and weekends. Testing during working days created a raw while testing during weekend resulted in error alert 'Booking of a concert is only done during working days'.

Form the examples 3 and 4 above, it has confirmed that trigger can be used to enforce complex security authorization as provided by security features of the database. For instance, disallowing booking of a concert during weekends: Assuming any concert should only hold during weekend. The concert should not hold in the month of August.

It is advisable to use BEFORE trigger for enforcing such a complex security authorization for these advantages:

i) the triggering statement is allowed to fire only after the security check, therefore, unauthorized statement would not lead to a wasted work
ii) ii) the triggering statement is only fired after the security check but not for each row affected by the triggering statement.

Example 5: Trigger creation to ensure that event records cannot be deleted if the date of the concert is not later than the current date:

```
CREATE OR REPLACE TRIGGER delete_old_record
        BEFORE DELETE ON event
        FOR EACH ROW
        BEGIN
                IF (:OLD.evt_date > sysdate) THEN
                        RAISE_APPLICATION_ERROR(-20012, 'No record is
                                allowed to be deleted if the event is yet be
                                        performed');
         END IF;
        END;
```

Eg5-Test1: DELETE FROM EVENT WHERE EVT_ID = 784;

Result: ERROR at line 1:

Eg5-Test2: DELETE FROM EVENT WHERE EVT_ID = 781;

Result: ERROR at line 1:

Eg5-Test1 resulted in error because the concert with event id 784 was not conducted. However, the concert with event id 781 in **Eg5-Test2** also resulted in error because of referential integrity relationship constraint. To be able to delete such record the foreign key constraint need to be deferred or the trigger should be created with delete cascade.

System trigger

Example 6: Creates a BEFORE statement trigger on the sample schema OPS$1223793. The database fires the trigger, when a user connected as OPS$1223793 and tries to drop a database object.

```
CREATE OR REPLACE TRIGGER dis_allow_drop_trigger
  BEFORE DROP ON OPS$1223793.SCHEMA
  BEGIN
```

```
    RAISE_APPLICATION_ERROR(
      num => -20004,
      msg => 'You are not permitted to drop object');
  END;
```

A conditional trigger

Example 7: Creates a DML trigger that uses conditional predicates to determine which of its possible triggering statements fired the trigger.

```
CREATE OR REPLACE TRIGGER conditional_Trigger
   BEFORE INSERT OR UPDATE OF concrt_id, evt_date
   OR DELETE ON event

 BEGIN
   CASE
        WHEN DELETING THEN
          DBMS_OUTPUT.PUT_LINE(You are deleting the arranged concert);
        WHEN INSERTING THEN
          DBMS_OUTPUT.PUT_LINE(You are inserting record on the event table);
    WHEN UPDATING('evt_date') THEN
          DBMS_OUTPUT.PUT_LINE(You are updating date of the event table);
        WHEN UPDATING('concrt_id') THEN
          DBMS_OUTPUT.PUT_LINE(You are updating concrt_id on the event table);
   END CASE;
END;
```

Multiple triggering statements are composed for the triggering event of a DML trigger. The conditional predicates deleting, inserting, or updating detects which triggering DLM statement should fire.

An INSTEAD OF trigger

Example: Creates INSTEAD OF trigger to show the update of a not inherently updatable view.

```
CREATE OR REPLACE VIEW about_conct As
  SELECT c.conct_id, c.conct_name, conct_duration
  c.conct_type, c.conct_cost, e.evt_id, e.evt_date
        FROM concert c, event e
         WHERE c.conct_id = e.conct_id;

CREATE OR REPLACE TRIGGER insert_about_conct
  INSTEAD OF INSERT ON about_conct
  DECLARE
    duplicate_info EXCEPTION;
    PRAGMA EXCEPTION_INIT (duplicate_info, -10002);
  BEGIN
  INSERT INTO concert
    (conct_id, conct_name, conct_duration, conct_type, conct_cost)
  VALUES  (:new.conct_id,
                :new.conct_name,
                :new.conct_duration,
                :new.conct_type,
```

```
            :new.conct_cost);
  INSERT INTO event
       (evt_id, evt_date, conct_id)
  VALUES(:new.evt_id,
          :new.evt_date,
          :new.conct_id);
  EXCEPTION
       WHEN duplicate_info THEN
            RAISE_APPLICATION_ERROR(
                 num=> -20116,
                 msg=> 'This is duplicate of concert and evt_id');
 END insert_about_conct;
```

The created view about_conct contains information about concert and arrangement of its event. The view is not inherently updatable (because the primary key of the EVENT table, evt_id, is not unique in the result set of the join view). The example creates an INSTEAD OF trigger to process *Insert* statements directed to the view. The trigger inserts rows into the base tables of the view, concert and event. As detailed in section 2.4, INSTEAD OF trigger is used to update the based table hence its views and raise an error message if the new insert is already in the table which is determined by the primary key.

Conclusion and Further Work

A database trigger is procedural code that is automatically executed in response to certain events on a particular table, view, schema, or the database. Various DBMSs do provide security solution; but some are more effective than others. In Oracle DBMS, triggers appear to be powerful tools in providing database security. The paper is aimed to implement, evaluate the application and the effectiveness of triggers in dealing with database security challenges especially for an organizational internal threats. Various applications of trigger techniques were used and evaluated such as collection of statistics for database access, auditing activities of database users, restriction of DML operations on table at unwanted period of time, prevention of invalid transaction, enforcement of different complex security authorization and raising issues for referential integrity. From the results of the empirical evaluation of applied, it has shown that proper implementation of database triggers could improve and maintain database security in a business operational environment.

The limitations of the research are that, firstly, it should have been implemented using the latest release of Oracle. Secondly, the researcher should have used the idea of ***views*** creation and allocated them to some users. The use of views could reduce the amount of policy creation. However, it would widen the research work.

References

Carl, D. (2012, December). OKUG: *Auditing techniques For Oracle Database 11g.* Uk Oracle User Group. Birmingham: ICC

Cheng, Z. (2012*).* Object-relational database approach for role-based access control (Rbac), *MSc. Dissertation*. USA: California. State University, Sacramento. Retrieved from http:// csusdspace.calstate.edu/bitstream/handle/10211.9/1731/

Fabbri, D., LeFevre, K. & Zhu, Q. D. (2011, September*) Policy replay: Misconfiguration response queries for data breach*: *the 36th VLDB International Conference on Very Large Data Bases*, Singapore: Proceedings of the Endowment, *3*(1)

Hall, T. (2020)**.** Trigger enhancements in Oracle Database 11g Release 1. Retrieved from https://oracle-base.com/articles/11g/trigger-enhancements-11gr1

IBM Cloud Education. (2019). Database Security: Learn the complexities of database security and some of the practices, policies, and technologies that will protect the confidentiality, integrity, and availability of your data. Retrieve from https://www.ibm.com/cloud/learn/database-security

Itai, Y.**,** Oludele, A.**,** & Goga, N. (2005). Trigger and database security. *International Journal of Computers & Technology*, *4*(1). Retrieve from DOI: 10.24297/ijct.v4i1b.3060

Lee, D., Mao, W., Chiu, H., & Chu, W. W. (2003). Designing triggers with trigger-by-example. *Under consideration for publication in Knowledge and Information Systems.*

Mazer, M. (2006). Auditing databases for compliance and risk management*. DM Review* 16(3), pp. 18. Retrieved from http://wlv.summon.serialssolutions.com

McAfee. (2012). Overview of McAfee real-time database monitoring, auditing, and intrusion prevention. *Compliance Week*, 8(94), pp. 60 Retrieved from www.mcafee.com/us/resources/.../wp-real-time-database-monitoring.pdf.

Noreen, Z., Hameed, I., & Usman, A. (2009). Development of database auditing infrastructure. *ACM*, pp.1-6. Retrieved from http://dl.acm.org/citation.cfm?id=1838092.

Oracle. (2020a). Database PL/SQL Language Reference. Retrieved from https://docs.oracle.com/cd/B28359_01/appdev.111/b28370/triggers.htm

Oracle. (2020b). Oracle 11g Oracle Database 11*g* Express Edition. *Release 2* (11.2) [online]. Retrieve from https://www.oracle.com/database/technologies/appdev/xe.html.

Oracle (2011). Oracle® 11*g* Database 2 Day + Security Guide: Auditing Database Activities *Release 1 (11.1).* Retrieved from http://docs.oracle.com/cd/B28359_01/server.111/b28337/tdpsg_auditing.htm

Qing, Z., Sheng, W. & Bin, L. (2010). Research and design of fine-grained permissions based on Quality of Detection and Management System. *IEEE*, pp. 331- 334. Retrieved from http://ieeexplore.ieee.org/xpl/3Farnumber%3D5608358.

Sheikh, S. (2017). Review paper on privacy and security of database management system. *Journal of Advanced Database Management & Systems*, 4(1), 21–24. Retrieved from http://computers.stmjournals.com/index.php?journal=JoADMS.

Yang, L. (2009, March). Teaching Database Security and Auditing,*SIGCSE'09:* Association for Computing Machinery's Special Interest Group on Computer Science Education, Chattanooga, TN, USA. *ACM.* Retrieved from http://www.academia.edu/24717193/Teaching_Database_Security_and_Auditing.