International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 07 Issue: 07 | July - 2023 SJIF Rating: 8.176 ISSN: 2582-3930

Understanding Database Triggers in Oracle: An Overview of Their Functionality, Utility, and Management

Sandeep Gupta

Abstract

A database trigger is a specialized block of PL/SQL code associated with a table operation like insertion, deletion, or updating of a row. When conditions specified within a trigger are met, it is automatically invoked, providing a layer of security by preventing potentially damaging operations on the database. Furthermore, triggers can cascade, meaning one trigger can instigate another. Despite the numerous benefits of triggers, their overuse can lead to complicated dependencies that may prove challenging to manage in expansive applications. Hence, it is prudent to employ triggers judiciously.

Keywords: Procedure, Security, Trigger.

Nomenclature:

SQL - Structured Query Language

PL/SQL - Procedural Language / Structured Query Language

DML - Data Manipulation Language

AN OVERVIEW OF DATABASE TRIGGERS

Stored within the database itself, database triggers are procedures implicitly set into motion by any changes made to a table. Unlike standard procedures that require explicit invocation by the user, triggers are automatically instigated by Oracle in response to table operations such as insertions, updates, or deletions, whether executed through SQL commands or an application.

For instance, consider a database application consisting of various SQL commands. When these commands are executed, they may implicitly instigate the firing of multiple triggers stored within the database. This implicit and automatic invocation of triggers under specific conditions highlights their unique nature in contrast to typical procedures [12].

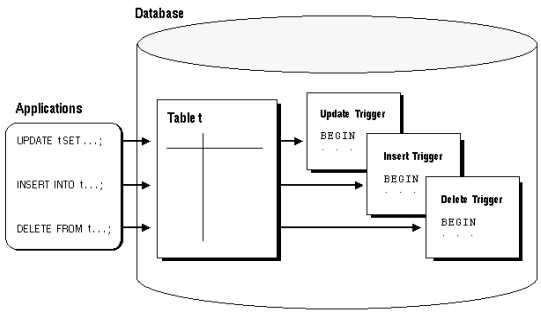

Fig. 1: Triggers

Triggers are distinct entities stored in the database, associated exclusively with tables rather than views. However, operations such as INSERT, UPDATE, or DELETE on the underlying tables of a view can activate triggers associated with those base tables.

UTILIZATION OF TRIGGERS

Triggers, in many instances, enhance the default capabilities of Oracle, resulting in a highly personalized database management system. One such use case involves permitting Data Manipulation Language (DML) operations on a table exclusively during conventional business hours. While Oracle's standard security features such as roles and privileges dictate who can initiate DML statements on the table, the trigger provides an additional layer of control by further limiting these operations to specific timeframes during business days. This represents a single instance of the potential for trigger-based customization within an Oracle database.

Triggers can also be employed for automatic generation of derived column values, prevention of invalid transactions, enforcement of intricate security authorizations, and the maintenance of referential integrity across nodes in a distributed database. Furthermore, triggers can be instrumental in enforcing complex business rules, transparent event logging, intricate auditing, maintaining synchronized table replicas, and collecting table access statistics.

ADVANTAGES OF TRIGGERS

Triggers offer various benefits:

• A trigger can restrict DML operations on a table to standard business hours or specific weekdays.

• It can aid in maintaining an audit trail of a table, preserving the records of modifications and deletions in the table, along with the operations performed and their timestamps.

• Triggers can prevent unauthorized transactions.

• They can enforce complex security authorizations.

SAFEGUARDING MEASURES WITH TRIGGERS

Certain aspects must be taken into account when working with triggers:

• The phenomenon of cascading can occur when a SQL statement within a trigger fires the same or another trigger. This sequence needs to be considered.

• Excessive customization of the database using triggers can lead to intricate interdependencies among triggers. This complexity may prove challenging to manage in large-scale applications.

DISTINGUISHING BETWEEN PROCEDURES AND TRIGGERS

• A key difference between a stored procedure and a trigger lies in their execution. While a stored procedure can be executed manually as needed using the exec command, a trigger is automatically invoked when an event (insert, delete, update) occurs on the table on which the trigger is defined.

• Stored procedures are versatile tools for performing user-specified tasks. They can accept parameters and return multiple result sets. On the other hand, triggers primarily serve auditing purposes. They enable tracking and recording of table events.

COMPONENTS OF A TRIGGER

• A Triggering Event or Statement: This is an SQL statement, such as insert, update, or delete, that initiates the firing of a trigger for a particular table. A triggering statement may also include multiple DML statements.

• A Trigger Restriction: A trigger restriction is a Boolean expression that must evaluate to TRUE for the trigger to fire. This feature is particularly applicable to triggers that fire for each row and serves to conditionally control the execution of a trigger. The WHEN clause is used to specify a trigger restriction.

• A Trigger Action: The procedure executed when a triggering statement is issued and the trigger restriction evaluates to TRUE is referred to as a trigger action. It may contain SQL and PL/SQL statements, define PL/SQL language constructs, and call stored procedures. Moreover, for row triggers, the statements in a trigger action can access the new and old column values of the current row being processed.

TRIGGER SYNTAX

The basic syntax for creating or replacing a trigger is as follows:

Create or replace trigger trigger-name {before, after}{delete, insert, update} on table-name

[referencing {OLD as old, NEW as new}] [for each row / statement [when condition]] declare

variable declaration;

constant declaration;

begin

pl/sql subprogram body; exception

exception pl/sql block;

end;

The 'create or replace' clause can be used to modify an existing trigger. The Fig. 2 diagram can be referenced for better understanding of the 'create trigger' statement. The OR REPLACE option recreates the trigger if it already exists, thereby allowing modification of an existing trigger's definition[13].

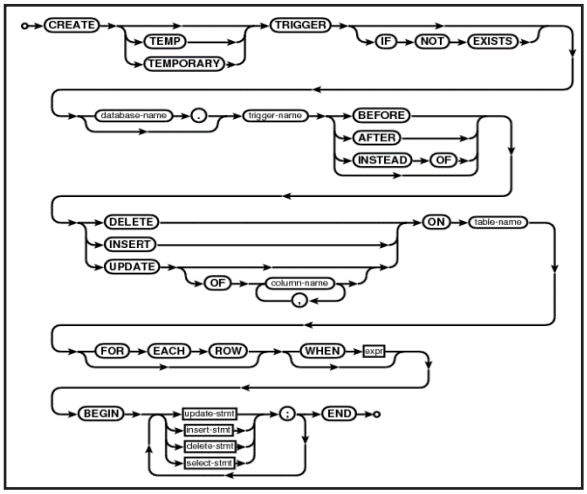

Fig. 2: Creation of Trigger

'BFORE' signifies that the trigger fires before the triggering statement is executed.

'AFTER' denotes that the trigger fires after the execution of the triggering statement.

'DELETE' signifies that the trigger is activated when a DELETE statement removes a row from the table.

'INSERT' denotes that the trigger is activated whenever an INSERT statement adds a row to the table.

'UPDATE' implies that the trigger is activated whenever an UPDATE statement modifies any column value in the table.

'ON' identifies the table on which the trigger will be created.

'REFERENCING' is used to define correlation names. In the PL/SQL block and WHEN clause of row triggers, correlation names allow for the specific referencing of old and new values of the current row. The default correlation names are OLD and NEW. If your row trigger is associated with a table named OLD or NEW, this clause allows you to assign different correlation names to avoid ambiguity between the table name and the correlation name.

'FOR EACH ROW' designates the trigger as a row trigger. A row trigger is fired once for every row affected by the triggering statement that also satisfies the optional trigger constraint defined in the WHEN clause. In the absence of this clause, the trigger is recognized as a statement trigger.

'WHEN' outlines the trigger restriction. The trigger restriction is a SQL condition that must be fulfilled for the trigger to fire. This condition must include correlation names and cannot comprise a query. You can specify a trigger restriction.

'Row Triggers and Statement Triggers' - When you establish a trigger, you decide how often the trigger action should be executed: it can be once for every row affected by the triggering statement (like an UPDATE statement that updates numerous rows), or it can be once for the entire triggering statement, irrespective of the number of affected rows.

'Row Triggers' - A row trigger executes each time the triggering statement impacts the table. For example, when an UPDATE statement changes multiple rows in a table, a row trigger fires once for each row that is affected. If a triggering statement does not impact any rows, a row trigger does not execute. Row triggers are useful when the code in the trigger action relies on data provided by the triggering statement or rows affected.

'Statement Triggers' - A statement trigger fires once on behalf of the triggering statement, regardless of the number of rows affected in the table, even if no rows are impacted. For example, if a DELETE statement removes several rows from a table, a statement-level DELETE trigger fires only once. Statement triggers are helpful when the code in the trigger action does not depend on data provided by the triggering statement or the rows affected. For instance, you can use a statement trigger to perform a complex security check on the current time or user or generate a single audit record.

'BEFORE and AFTER Triggers' - When setting up a trigger, you can specify the trigger timing - whether the trigger action should be run before or after the triggering statement. 'BEFORE' and 'AFTER' apply to both statement and row triggers. 'BEFORE' and 'AFTER' triggers fired by DML statements can only be defined on tables, not on views. However, if an INSERT, UPDATE, or DELETE statement is issued against a view, triggers on the base tables of the view are fired. 'BEFORE' and 'AFTER' triggers fired by DDL statements can only be defined on the database or a schema, not on specific tables.

'BEFORE Triggers' - 'BEFORE' triggers execute the trigger action prior to the triggering statement. This type of trigger is often used when the trigger action determines whether the triggering statement should be allowed to complete. With a 'BEFORE' trigger, you can avoid unnecessary processing of the triggering statement and its eventual rollback in cases where an exception is raised in the trigger action. It's also used to derive specific column values before executing a triggering INSERT or UPDATE statement.

'AFTER Triggers' - 'AFTER' triggers execute the trigger action after the triggering statement is run.

'INSTEAD OF Triggers' - These triggers offer an unobtrusive method of altering views that are not directly modifiable via DML operations (INSERT, UPDATE, and DELETE). They are named 'INSTEAD OF' triggers because, unlike other types of triggers, Oracle initiates the trigger action instead of performing the triggering statement.

'Trigger Type Combinations' - Different types of triggers can be blended, resulting in combinations like 'before statement trigger', 'before row trigger', and so on.

'Cascading Triggers' - When a trigger is activated, the SQL statements within that trigger can potentially activate another trigger. This is demonstrated in Fig. 3. When a statement within a trigger's body results in the activation of another trigger, these triggers are said to be cascading[12].

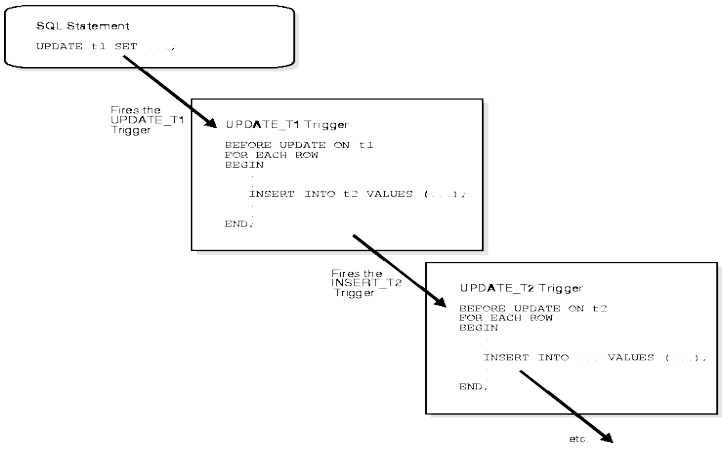

Fig. 3 Trigger Cascading

While the use of triggers can effectively customize a database, they should be utilized sparingly. Excessive reliance on triggers can lead to intricate interdependencies, which could become challenging to manage in a large-scale application.

'Practical Application of Triggers' -

We start by constructing a 'sailor' table as shown below:

CREATE TABLE sailor(sid NUMBER(3), fname VARCHAR2(30), rating NUMBER(2), age NUMBER(3));

Next, we introduce a few records into the 'sailor' table:

INSERT INTO sailor VALUES(1,'veer', 5, 20);

INSERT INTO sailor VALUES(2,'vinay', 7, 22);

INSERT INTO sailor VALUES(3,'raja', 15, 18);

Now, if we display the 'sailor' table, it appears as follows:

SELECT * FROM sailor;

SID	SNAME	RATING	AGE
1	Veer	5	20
2	Vinay	7	22
3	Raja	15	18

Next, we create another table, 'sailor_history', as follows:

CREATE TABLE sailor_history(sid NUMBER(3), sname VARCHAR2(30));

We then establish a trigger called 'sailor_delete', which will maintain a historical record of sailors removed from the 'sailor' table. These records are stored in the 'sailor_history' table.

CREATE OR REPLACE TRIGGER sailor_delete

AFTER DELETE ON sailor

FOR EACH ROW

BEGIN

```
  INSERT INTO sailor_history VALUES(:OLD.SID, :OLD.SNAME);
  DBMS_OUTPUT.PUT_LINE('Record successfully removed from sailor table and stored in sailor_history table');
END;
```

Now, if we delete a record from the 'sailor' table:

```
DELETE FROM sailor WHERE sid=1;
```

The 'sailor_delete' trigger would be activated and provide the following output while altering the two tables:

'Record successfully removed from sailor table and stored in sailor_history table'

'1 row(s) deleted.'

Now, we can observe how the tables have been altered:

select * from sailor; gives us,

SID	SNAME	RATING	AGE
2	Vinay	7	22
3	Raja	15	18

select * from sailor_history; gives us,

SID	SNAME
1	Veer

Here are some additional trigger examples:

1. Establish a database trigger that prevents transactions on the 'route_detail' table from being carried out on Saturdays and Sundays. The trigger is activated before each row insertion.

The 'route_detail' table is structured as follows: (rid, from, to, shipping_date)

```
CREATE OR REPLACE TRIGGER sun_trig
BEFORE INSERT OR UPDATE OR DELETE ON route_detail
DECLARE
  shipping_day CHAR;
BEGIN
  shipping_day := TO_CHAR(SYSDATE, 'DY');
  IF shipping_day IN ('SAT', 'SUN') THEN
raise_application_error(-20001, ‘try on anyweekdays’);
end if;
end;
```

This trigger, 'sun_trig', uses the system date to determine the day of the week and restricts transactions from occurring on the weekend.

Let's craft a trigger that fires before the execution of a DML statement on the 'employees' table. This trigger verifies the current day and time, based on sysdate. If it's Sunday, or if the current time is outside normal business hours (8:00 a.m. to 5:00 p.m.), the trigger prevents the execution of the DML statement and raises an exception with a relevant message.

Here's how you might define such a trigger:

```
CREATE OR REPLACE TRIGGER day_chk
BEFORE INSERT OR UPDATE OR DELETE ON employees
DECLARE
 msg EXCEPTION;
BEGIN
 IF TO_CHAR(SYSDATE,'DAY') = 'SUNDAY' OR
    TO_CHAR(SYSDATE,'HH24:MI') < '08:00' OR
    TO_CHAR(SYSDATE,'HH24:MI') > '17:00'
 THEN
   RAISE msg;
 END IF;
EXCEPTION
 WHEN msg THEN
   DBMS_OUTPUT.PUT_LINE('Changes to the employees table are only permitted during business hours.');
END;
```

In this script, the 'day_chk' trigger is created. It examines the current day and time, and if the conditions are met, it raises a custom exception. This exception is then caught in the EXCEPTION block and a message is printed, explaining that changes to the employees table are only allowed during business hours.

Let's create a row-level trigger that runs before an insert or update operation on the 'route_header' table. This trigger will ensure that the distance between the origin and destination does not exceed 2000 km.

Here's how such a trigger could be defined:

```
CREATE OR REPLACE TRIGGER check_distance
BEFORE INSERT OR UPDATE ON route_header
FOR EACH ROW
DECLARE
 excessive_distance EXCEPTION;
 route_distance route_header.distance%TYPE;
BEGIN
 SELECT distance INTO route_distance FROM route_header WHERE route_id = :NEW.route_id;
 IF route_distance > 2000 THEN
   RAISE excessive_distance;
 END IF;
EXCEPTION
 WHEN excessive_distance THEN
   DBMS_OUTPUT.PUT_LINE('The distance between origin and destination cannot exceed 2000 km.');
END;
```

In this script, the 'check_distance' trigger is created. It retrieves the distance for the new or updated row from the 'route_header' table. If the distance is greater than 2000 km, it raises a custom exception. This exception is then caught in

the EXCEPTION block and a message is printed, informing that the distance between origin and destination should not exceed 2000 km.

Please note that this script assumes the 'route_header' table has a 'distance' column, which wasn't listed in your table schema. Make sure you adjust it to your actual data model.

A database trigger can exist in one of two states: enabled or disabled. When you disable a trigger, it can lead to performance improvements, particularly when substantial modifications are made to the table data since the trigger logic is not executed.

Consider the following example: `UPDATE emp SET name = UPPER(name);`. This command may run faster if all triggers activated by updates on the 'emp' table are disabled.

We can use the ALTER command to switch a trigger's state between enabled and disabled. Here's how it works:

To disable a specific trigger, use the following command: `*ALTER TRIGGER trigger_name DISABLE*;`. For instance, to disable the 'sun_trig' trigger, you would write: `*ALTER TRIGGER sun_trig DISABLE*;`.

To enable a disabled trigger, use the following command: `*ALTER TRIGGER trigger_name ENABLE*;`. To enable the 'sun_trig' trigger again, you would use: `*ALTER TRIGGER sun_trig ENABLE*;`.

If you want to disable all triggers associated with a certain table, you can use the following command: `*ALTER TABLE table_name DISABLE ALL TRIGGERS*;`. For instance, to disable all triggers on the 'emp' table, you would write: `*ALTER TABLE emp DISABLE ALL TRIGGERS*;`.

Triggers can be removed when they are no longer necessary. This is done using the **'DROP TRIGGER'** command, as illustrated below:
`*DROP TRIGGER trigger_name*;`
For instance, if you want to remove the 'sun_trig' trigger, you would write:
`*DROP TRIGGER sun_trig*;`

To view detailed information about a specific trigger, you can use SQL queries on the system view 'USER_TRIGGERS'. For example, to see the type, triggering event, and associated table of a trigger named 'PART', you would use the following command:

`*SELECT Trigger_type, Triggering_event, Table_name FROM USER_TRIGGERS WHERE Trigger_name = 'PART';*`

The result may look something like this:

TRIGGER_TYPE	TRIGGERING_EVENT	TABLE_NAME
AFTER STATEMENT	INSERT	PART

To view the body of the 'PART' trigger, use the following command:

`SELECT Trigger_body FROM USER_TRIGGERS WHERE Trigger_name = 'PART';`

The result might be similar to the following:

| TRIGGER_BODY |

| ---------- |
| BEGIN... |
| END; |

In this example, 'BEGIN...' and 'END;' represent the beginning and end of the trigger's code block. For instance, you might see a command like 'DBMS_OUTPUT.PUT_LINE('The insert in part table is successful');' in the output. This command would output the specified message whenever the trigger is activated.

CONCLUSION

Database triggers serve as valuable tools for enforcing intricate business rules that exceed the capabilities of standard integrity constraints. They can also provide utility in logging activities and automating certain actions. Triggers can be activated either before or after a data manipulation language (DML) operation, and they can be set to respond either to each row affected by the operation or to the operation as a whole.

INSTEAD-OF triggers offer a unique functionality; they activate in place of the specified DML command. These triggers are typically defined on relational views or object views.

There may be scenarios where a trigger could be temporarily disabled to improve performance, especially during large-scale data manipulation on the table. Once the heavy data processing is complete, the trigger can be enabled again.

Furthermore, triggers contribute to data security. If any condition is violated, the trigger will automatically activate, potentially preventing unauthorized or harmful operations.

REFERENCES

[1] *A. Behrend, C. Dorau, and R. Manthey, "SQL triggers reacting on time events: An extension proposal," in J. Grundspenkis, T. Morzy, and G. Vossen, (eds), Advancesin Databases and Information Systems (ADBIS 2009),Lecture Notes in Computer Science, vol. 5739, Springer, Berlin, Heidelberg, 2009.*

[2] *L. Pamulaparty, T. P. Kumar, and P. V. B. Varma, "A survey: Security perspectives of ORACLE and IBM-DB2 databases," International Journal of Scientific and Research Publications, vol. 3, no. 1, January 2013.*

[3] *D. Lee, W. Mao, H. Chiu, and W. W. Chu, "Designing triggers with trigger-by-example," Knowledge and Information Systems, vol. 7, no. 1, pp. 110-134, 2005.*

[4] *J. J. King, "Oracle 11g/10g for developers: What you needto know," Session S300195, King trainging resources, 2008.*

[5] *An Oracle white paper, "Oracle Database Security Checklist," June 2008.*

[6] *Kornbrust, "Best Practices for Oracle DatabasesHardening Oracle 10.2.0.3 / 10.2.0.4," N. D.*

[7] *Bayross, SQL, PL/SQL the Programming Language ofOracle, N. D.*

[8] *Oracle for beginners, "Ch-20 Database Triggers," N. D.srikanthtechnologies.com*

[9] *N. Shah "RDBMS," Mahajan Publication, 2011.*

[10] *"Oracle," Atmiya Infotech Pvt Ltd.*

[11] *"PL/SQL Triggers,". Available: http://docs.oracle.com/cd/E11882_01/appdev.112/e25519/triggers.htm.*

12] *"Database Triggers - Oracle7 Server Concepts Manual,".Available: ocs.oracle.com*

/cd/A57673_01/DOC/server/ doc/SCN73/ch15.htm?.

[13] *"Create Trigger,". Available: http://www.sqlite.org/lang_createtrigger.html*