TRIGGERS AND ROUTINES

Triggers and routines are very powerful database objects because they are stored in the database and controlled by the DBMS. Thus, the code required to create them is stored in only one location and is administered centrally. As with table and column constraints, this promotes stronger data integrity and consistency of use within the database; it can be useful in data auditing and security to create logs of information about data updates. Not only can triggers be used to prevent unauthorized changes to the database, they can also be used to evaluate changes and take actions based on the nature of the changes.
A significant advantage of a trigger over a constraint to accomplish the same control is that the processing logic of a trigger can produce a customized user message about the occurrence of a special event, whereas a constraint will produce a standardized, DBMS error message, which often is not very clear about the specific event that occurred.

Triggers: A named set of SQL statements that are considered (triggered) when a data modification (i.e., INSERT, UPDATE, DELETE) occurs or if certain data definitions are encountered. If a condition stated within a trigger is met, then a prescribed action is taken.

- Triggers can also cascade, causing other triggers to fire.
- Triggers can be used to ensure referential integrity, enforce business rules, create audit trails, replicate tables, or activate a procedure
- Triggers are used when you need to perform, under specified conditions, a certain action as the result of some database event (e.g., the execution of a DML statement such as INSERT, UPDATE, or DELETE or the DDL statement ALTER TABLE).
- A trigger has three parts: event, condition, and action.
- Example:
  Consider the following example from Pine Valley Furniture Company: Perhaps the manager in charge of maintaining inventory needs to know (the action of being informed) when an inventory item's standard price is updated in the Product_T table (the event). After creating a new table, PriceUpdates_T, a trigger can be written that enters each product when it is updated, the date that the change was made, and the new standard price that was entered. The trigger is named StandardPriceUpdate, and the code for this trigger follows:

```
CREATE TRIGGER StandardPriceUpdate
AFTER UPDATE OF ProductStandardPrice ON Product_T
FOR EACH ROW
INSERT INTO PriceUpdates_T VALUES (ProductDescription, SYSDATE,
ProductStandardPrice);
```

  - In this trigger, the event is an update of ProductStandardPrice, the condition is FOR EACH ROW (i.e., not just certain rows), and the action after the event is to insert the specified values in the PriceUpdates_T table, which stores a log of when (SYSDATE) the change occurred and important information about changes made to the ProductStandardPrice of any row in the table.
  - In the case just shown, the trigger should insert the new standard price information into PriceUpdate_T after Product_T has been updated.
- Triggers may occur either before, after, or instead of the statement that aroused the trigger is executed.
  - An "instead of" trigger is not the same as a before trigger but executes instead of the intended transaction, which does not occur if the "instead of" trigger fires.
- DML triggers may occur on INSERT, UPDATE, or DELETE commands.
  - And they may fire each time a row is affected, or they may fire only once per statement, regardless of the number of rows affected.
- DDL triggers are useful in database administration and may be used to regu late database operations and perform auditing functions.
  - They fire in response to DDL events such as CREATE, ALTER, DROP, GRANT, DENY, and REVOKE.
  - Example:

```
CREATE TRIGGER safety
ON DATABASE
FOR DROP_TABLE, ALTER_TABLE
AS
    PRINT 'You must disable Trigger "safety" to drop or alter tables!'
    ROLLBACK;
```

Routines: In contrast to triggers, which are automatically run when a specified event occurs, routines must be explicitly called, just as the built-in functions (such as MIN and MAX) are called.

- The routines have been developed to address shortcomings of SQL as an application development language—originally, SQL was only a data retrieval and manipulation language. Therefore, SQL is still typically used in conjunction with computationally more complete languages, such as traditional 3G languages (e.g., Java, C#, or C) or scripting languages (e.g., PHP or Python), to create business applications, procedures, or functions.
- The extensions that make SQL computationally complete include flow control capabilities, such as IF-THEN, FOR, WHILE statements, and loops, which are contained in a package of extensions to the essential SQL specifications.
- Routines or in other words user made sql functions are stored as persistent stored modules because they are essentially object or pieces of code that are not dropped unless you explicitly drop them thus they can be used over and over again.
- SQL/PSM includes several SQL control statements:

STATEMENT	DESCRIPTION
CASE	Executes different sets of SQL sequences, according to a comparison of values or the value of a WHEN clause, using either search conditions or value expressions. The logic is similar to that of an SQL CASE expression, but it ends with END CASE rather than END and has no equivalent to the ELSE NULL clause.
IF	If a predicate is TRUE, executes an SQL statement. The statement ends with an ENDIF and contains ELSE and ELSEIF statements to manage flow control for different conditions.
LOOP	Causes a statement to be executed repeatedly until a condition exists that results in an exit.
LEAVE	Sets a condition that results in exiting a loop.
FOR	Executes once for each row of a result set.
WHILE	Executes as long as a particular condition exists. Incorporates logic that functions as a LEAVE statement.
REPEAT	Similar to the WHILE statement but tests the condition after execution of the SQL statement.
ITERATE	Restarts a loop.

- A function returns one value and has only input parameters. You have already seen the many built-in functions included in SQL, including the newest functions listed in Table 6-1. \
- A procedure may have input parameters, output parameters, and parameters that are both input and output parameters.

The following are some of the advantages of SQL-invoked routines:

- ***Flexibility*** Routines may be used in more situations than constraints or triggers, which are limited to data-modification circumstances. Just as triggers have more code options than constraints, routines have more code options than triggers.
- ***Efficiency*** Routines can be carefully crafted and optimized to run more quickly than slower, generic SQL statements.
- ***Sharability*** Routines may be cached on the server and made available to all users so that they do not have to be rewritten.
- ***Applicability*** Routines are stored as part of the database and may apply to the entire database rather than be limited to one application. This advantage is a corollary to sharability.

-

- Example of a routine:

To build a simple procedure that will set a sale price, the existing Product_T table in Pine Valley Furniture Company is altered by adding a new column, SalePrice, that will hold the sale price for the products:

```
ALTER TABLE Product_T
ADD (SalePrice DECIMAL (6,2));
```

This is a simple sql code

Result:

```
Table altered.
```

- We will now make this sql code into a routine:

```
CREATE OR REPLACE PROCEDURE ProductLineSale
  AS BEGIN
    UPDATE Product_T
      SET SalePrice =.90 * ProductStandardPrice
      WHERE ProductStandardPrice > = 400;
    UPDATE Product_T
      SET SalePrice =.85 * ProductStandardPrice
      WHERE ProductStandardPrice < 400;
END;
```

The procedure scans all rows of the Product_T table. Products with a ProductStandardPrice of $400 or higher are discounted 10 percent, and products with a ProductStandardPrice of less than $400 are discounted 15 percent.

Oracle returns the comment "Procedure created" if the syntax has been accepted.

To run the procedure in Oracle, use this command (which can be run interactively, as part of an application program, or as part of another stored procedure):

```
SQL > EXEC ProductLineSale
```

Oracle gives this response:

```
PL/SQL procedure successfully completed.
```