Triggers are stored programs, which are automatically executed or fired when some events occur. Triggers are, in fact, written to be executed in response to any of the following events:

- A database manipulation *DML statement* `DELETE, INSERT, OR UPDATE`.
- A database definition *DDL statement* `CREATE, ALTER, OR DROP`.
- A database operation `SERVERERROR, LOGON, LOGOFF, STARTUP, OR SHUTDOWN`.

Triggers could be defined on the table, view, schema, or database with which the event is associated.

**Benefits of Triggers**

Triggers can be written for the following purposes:

- Generating some derived column values automatically
- Enforcing referential integrity
- Event logging and storing information on table access
- Auditing
- Synchronous replication of tables
- Imposing security authorizations
- Preventing invalid transactions

**Creating Triggers**

The syntax for creating a trigger is:

```sql
CREATE [OR REPLACE] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF}
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
on table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
DECLARE
    Declaration-statements
BEGIN
    Executable-statements
EXCEPTION
    Exception-handling-statements
END;
```

Where,

- `CREATE [OR REPLACE] TRIGGER trigger_name`: Creates or replaces an existing trigger with the `trigger_name`.
- `{BEFORE | AFTER | INSTEAD OF}`: This specifies when the trigger would be executed. The `INSTEAD OF` clause is used for creating trigger on a view.
- `{INSERT [OR] | UPDATE [OR] | DELETE}`: This specifies the DML operation.
- `[OF col_name]`: This specifies the column name that would be updated.
- **[ON table_name]**: This specifies the name of the table associated with the trigger.

- **[REFERENCING OLD AS o NEW AS n]**: This allows you to refer new and old values for various DML statements, like INSERT, UPDATE, and DELETE.

- **[FOR EACH ROW]**: This specifies a row level trigger, i.e., the trigger would be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.

- **WHEN condition**: This provides a condition for rows for which the trigger would fire. This clause is valid only for row level triggers.

### Example:

To start with, we will be using the CUSTOMERS table we had created and used in the previous chapters:

```
Select * from customers;
```

<table>
<thead>
<tr>
<th>ID</th>
<th>NAME</th>
<th>AGE</th>
<th>ADDRESS</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ramesh</td>
<td>32</td>
<td>Ahmedabad</td>
<td>2000.00</td>
</tr>
<tr>
<td>2</td>
<td>Khilan</td>
<td>25</td>
<td>Delhi</td>
<td>1500.00</td>
</tr>
<tr>
<td>3</td>
<td>kaushik</td>
<td>23</td>
<td>Kota</td>
<td>2000.00</td>
</tr>
<tr>
<td>4</td>
<td>Chaitali</td>
<td>25</td>
<td>Mumbai</td>
<td>6500.00</td>
</tr>
<tr>
<td>5</td>
<td>Hardik</td>
<td>27</td>
<td>Bhopal</td>
<td>8500.00</td>
</tr>
<tr>
<td>6</td>
<td>Komal</td>
<td>22</td>
<td>MP</td>
<td>4500.00</td>
</tr>
</tbody>
</table>

The following program creates a **row level** trigger for the customers table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old values and new values:

```
CREATE OR REPLACE TRIGGER display_salary_changes
BEFORE DELETE OR INSERT OR UPDATE ON customers
FOR EACH ROW
WHEN (NEW.ID > 0)
DECLARE
    sal_diff number;
BEGIN
    sal_diff := :NEW.salary - :OLD.salary;
    dbms_output.put_line('Old salary: ' || :OLD.salary);
    dbms_output.put_line('New salary: ' || :NEW.salary);
    dbms_output.put_line('Salary difference: ' || sal_diff);
END;
/
```

When the above code is executed at SQL prompt, it produces the following result:

```
Trigger created.
```

Here following two points are important and should be noted carefully:

- OLD and NEW references are not available for table level triggers, rather you can use them for record level triggers.

- If you want to query the table in the same trigger, then you should use the AFTER keyword, because triggers can query the table or change it again only after the initial changes are applied and the table is back in a consistent state.

- Above trigger has been written in such a way that it will fire before any DELETE or INSERT or UPDATE operation on the table, but you can write your trigger on a single or multiple operations, for example BEFORE DELETE, which will fire whenever a record will be deleted using DELETE operation on the table.
**Triggering a Trigger**

Let us perform some DML operations on the CUSTOMERS table. Here is one INSERT statement, which will create a new record in the table:

```sql
INSERT INTO CUSTOMERS (ID, NAME, AGE, ADDRESS, SALARY)
VALUES (7, 'Kriti', 22, 'HP', 7500.00);
```

When a record is created in CUSTOMERS table, above create trigger `display_salary_changes` will be fired and it will display the following result:

- **Old salary:**
- **New salary:** 7500
- **Salary difference:**

Because this is a new record so old salary is not available and above result is coming as null. Now, let us perform one more DML operation on the CUSTOMERS table. Here is one UPDATE statement, which will update an existing record in the table:

```sql
UPDATE customers
SET salary = salary + 500
WHERE id = 2;
```

When a record is updated in CUSTOMERS table, above create trigger `display_salary_changes` will be fired and it will display the following result:

- **Old salary:** 1500
- **New salary:** 2000
- **Salary difference:** 500