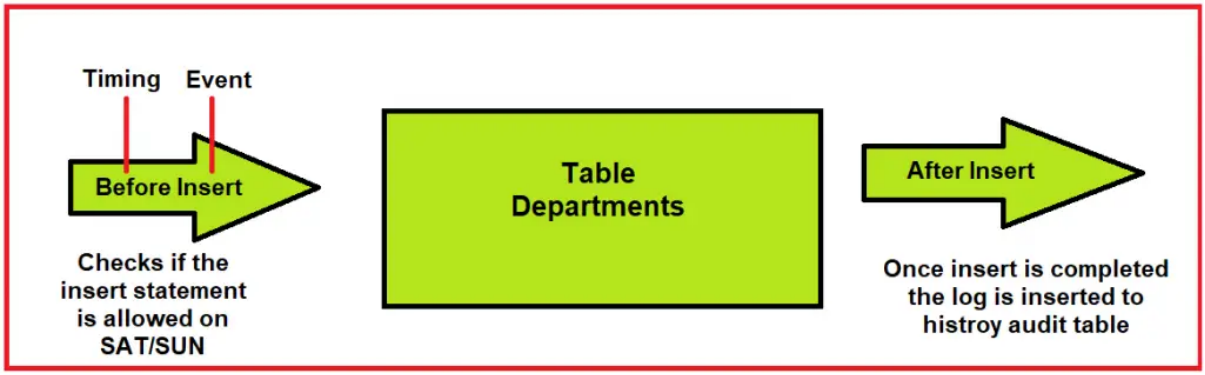
Triggers in Oracle

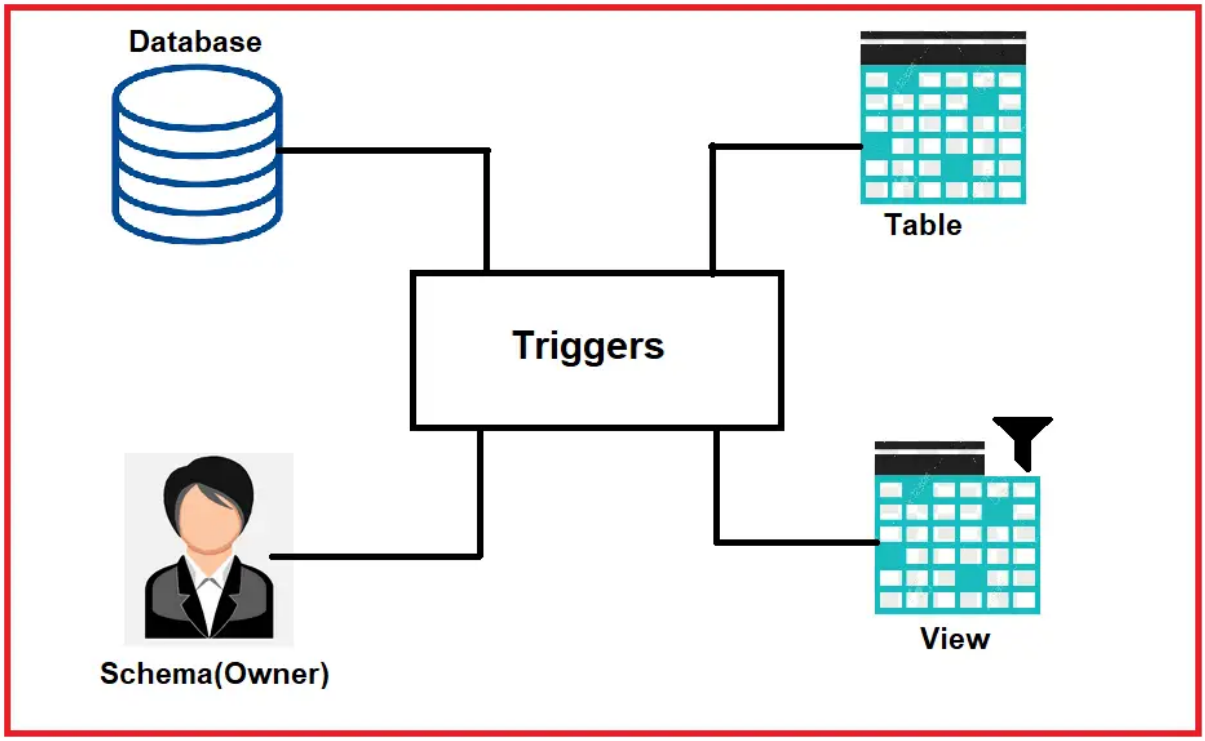
What are Triggers in Oracle?

A Trigger is PL/SQL block that is stored in the database and fired(executed) in response to a specific event. The oracle database automatically executes a trigger when specified conditions occur. This is a schema object like procedures and functions. We can create a trigger and oracle executes the trigger based on the specific conditions.



Defining Triggers in Oracle:

Triggers can be defined on the table, view, schema, or database.



Trigger Event Types:

We can create and execute triggers whenever one of the following operations occurs in the database.

1. A database manipulation (DML) statement (DELETE, INSERT, or UPDATE).
2. A database definition (DDL) statement (CREATE, ALTER, DROP).
3. A database operation such as SERVER ERROR, LOG ON, LOG OFF, STARTUP, or SHUTDOWN.

Triggers can be distinguished into two types.

1. **Database Triggers:** The triggers will execute whenever a DML, DDL, or system event occurs on a schema or a database.
2. **Application Triggers:** The trigger will execute whenever an event occurs within a particular application.

Uses of Security:

1. **Security:** We can create a trigger in such a way that no user cannot run an insert statement on holidays. This will help the management to increase security.

2. **Auditing:** We can create a trigger to log all the transactions that happened on a particular table. This is helpful for a DBA to track the transactions
   that occurred in the database.
3. **Data Integrity:** We can create a trigger to make sure to follow the standard rules.
4. **Referential Integrity:** we can make triggers to make sure the foreign key and primary key are
   in case the objects in the database do not have many relations. In some cases, we may find that there will be no relations on the tables in the database but there are relations at the application level. We can check the relations using triggers.
5. **Table Replication:** We can create a trigger in order to synchronize a table. We use
   triggers for computing derived data automatically and event logging.

Trigger Types in Oracle:

We have many triggers types

1. **Simple DML Triggers:** The DML triggers are commonly used triggers. Users can be created
   BEFORE, AFTER, INSTEAD OF. Before inserting, update a specific table. After updating a specific table. Instead of is used in views.
2. **Compound Triggers:** These triggers will be covered later.
3. **Non- DML Triggers:** The DDL event triggers occurs whenever the user creates a table or
   creates a view. Database event triggers occur whenever there is an event occurred in the database. Suppose, if a user loggedin to the database, then a trigger is created and executed to log all the information that the user executes in the database.

Trigger Event Types and Body:

A trigger event type determines which DML statement causes the trigger to execute. The possible events are:

1. Insert
2. Update(column)

3. Delete

DML statement event indicates whenever an insert statement is executed on a specific table or an update statement or delete statement is executed.
The update is specified in the column as well as a single column is updated in a specific table. A trigger body determines what action needs to be performed and is a PL/SQL or a call to a procedure.

DML Triggers in Oracle

DML Triggers in Oracle

The DML Triggers are the Data Manipulation triggers that make changes to the table inside the database. There are two types of DML triggers.

1. Statement Level DML Triggers
2. Row Level DML Triggers

Statement Level Triggers

1. This is the default trigger while creating a trigger.
2. Executes only once for the triggering event.
3. Executes only once even if no rows are affected.
4. Example: Security check on the user, time, etc.

Row Level Triggers

1. Use the FOR EACH ROW clause when creating a trigger.
2. Fires once for each row affected by the triggering event.
3. Does not fire if the triggering event does not affect any rows.
4. Example: Log the transactions.

```
Create table test
(emp_id number,
 ename varchar2(100)
);

Insert into test values (1,'abc');
Insert into test values (2,'def');
```

```
Create or replace trigger test_before_update
Before update
On test
Begin
DBMS_OUTPUT.PUT_LINE ('TRIGGER EXECUTED SUCCESSFULLY');
End;



Update test set ename=ename||'s';

Update test set ename=ename||'sahoo' Where emp_id=20;
```

```
Create or replace trigger test_before_update
Before update
On test
For each row
Begin
DBMS_OUTPUT.PUT_LINE ('TRIGGER EXECUTED SUCCESSFULLY');
End;

Update test set ename=ename||'s';
Update test set ename=ename||'sahoo' Where emp_id=20;
```

Firing Point: BEFORE

- BEFORE INSERT TRIGGER
- BEFORE UPDATE TRIGGER
- BEFORE DELETE TRIGGER

Firing Point: AFTER

- AFTER INSERT TRIGGER
- AFTER UPDATE TRIGGER
- AFTER DELETE TRIGGER

Oracle Before INSERT/UPDATE/DELETE Trigger

This statement specifies that Oracle will fire this trigger **BEFORE** the INSERT/UPDATE or DELETE operation is executed.

Limitations

- BEFORE trigger cannot be created on a view.

- You cannot update the OLD values.
- You can only update the NEW values.

Statement Level Trigger

```
DROP Table Department;
CREATE TABLE Department (
  DepartmentId INT,
  DepartmentName VARCHAR(15)
);

INSERT INTO Department (DepartmentId, DepartmentName) VALUES (10, 'IT');
INSERT INTO Department (DepartmentId, DepartmentName) VALUES (20, 'HR');
INSERT INTO Department (DepartmentId, DepartmentName) VALUES (30, 'Finance');
```

```
delete from departments;
```

```
Create or replace trigger dept_check_time
Before
Insert or update or delete
On departments
Begin
 If to_number(to_char(sysdate,'hh224')) not between 7 and 15 then
 Raise_application_error(-0010,'DML Operations not allowed now');
 End if;
End;
```

```
Delete from departments;
```

```
Delete from departments where department_id = -3;
```

```
select * from user_objects where object_name='DEPT_CHECK_TIME';
```

```
select * from user_triggers where trigger_name='DEPT_CHECK_TIME';
```

```
Create or replace trigger dept_check_time
Before
Insert or update or delete
On departments
Begin
 If to_number(to_char(sysdate,'hh224')) not between 7 and 15 then
 If inserting then
 Raise_application_error(-20010,'Insert Operations not allowed now');
 elseif deleting then
 Raise_application_error(-20010,'Delete Operations not allowed now');
 elseif updating then
 Raise_application_error(-20010,'Update Operations not allowed now');
 End if;
End if
End;
```

```
Delete from departments;
```

Row Level Trigger

```
Create table COPY_EMP as select * from employee;
```

```
Select * from COPY_EMP;
```

```
Create or replace trigger check_salary
Before
Insert or update of salary
On COPY_EMP
For each row
Begin
 If :new.salary<50000 then
 Raise_application_error(-20030,'minimum salary is 50000');
 End if;
End;



Update COPY_EMP Set salary=20000 Where employee_id=1001;
```

```
select count(1) from copy_emp;
```

```
Update COPY_EMP Set salary=200;
```

```
Create or replace trigger check_salary
Before
Insert or update of salary
On COPY_EMP
REFERENCING NEW AS NEW OLD AS OLD

For each row
Begin
 If :new.salary<50000 then
 Dbms_output.put_line('TRIGGER EXECUTED SUCCESSFULLY');
 End if;
End;







Update COPY_EMP Set salary=20000 Where employee_id=1001;

Update copy_emp Set salary=200;
```

```
Create or replace trigger check_salary
Before
Insert or update of salary
On COPY_EMP
REFERENCING NEW AS N OLD AS O
For each row
Begin
 If :n.salary<500 then
 Dbms_output.put_line('TRIGGER EXECUTED SUCCESSFULLY');
 End if;
End;


Update COPY_EMP Set salary=20000 Where employee_id=1001;
```

Audit Table using Triggers

`drop table copy_emp;`

`Create table copy_emp as select * from employee;`

`select * from copy_emp;`

```
CREATE TABLE EMP_SALARY_AUDIT
(
  EMP_ID NUMBER
  OPERATION VARCHAR2(100),
  OLD_SAL NUMBER,
  NEW_SAL NUMBER,
  OP_DATE DATE,
  BY_USER VARCHAR2(100)
);
```

```
create or replace trigger emp_copy_sal_audit
after insert or update of salary or delete
on copy_emp
for each row
begin
  if inserting then
  insert into EMP_SALARY_AUDIT(EMP_ID, OPERATION,OLD_SAL, NEW_SAL, OP_DATE, BY_USER)
  values (:new.employee_id,'Inserting',null,:new.salary,sysdate,user);
  end if;

  if updating then
  insert into EMP_SALARY_AUDIT(EMP_ID,OPERATION,OLD_SAL,NEW_SAL,OP_DATE,BY_USER)
  values (:old.employee_id,'updating',:old.salary,:new.salary,sysdate,user);
  end if;

  if deleting then
  insert into EMP_SALARY_AUDIT(EMP_ID,OPERATION,OLD_SAL,NEW_SAL,OP_DATE,BY_USER)
  values (:old.employee_id,'deleting',:old.salary,null,sysdate,user);
```

```
  end if;
end;
```

```
insert into copy_emp (EMPLOYEE_ID, FIRST_NAME, LAST_NAME, EMAIL, salary, hire_date,
 job_id)
values (1,'JAY','JAY','dotnet@hotmail.com',200,sysdate,'ORACLE_DBA');

Select * from employees where employee_id=1;
```

```
select EMP_ID, OPERATION, OLD_SAL, NEW_SAL,
to_char(OP_DATE,'dd-mm-yyyy hh24:mi:ss') OP_DATE , BY_USER
from EMP_SALARY_AUDIT
order by OP_DATE;
```

```
Update copy_emp Set salary=salary+10 Where department_id=20;
```

```
select EMP_ID, OPERATION, OLD_SAL, NEW_SAL,
to_char(OP_DATE,'dd-mm-yyyy hh24:mi:ss') OP_DATE , BY_USER
from EMP_SALARY_AUDIT
order by OP_DATE;
```

```
Delete from copy_emp Where employee_id=1;
```

```
select EMP_ID, OPERATION, OLD_SAL, NEW_SAL,
to_char(OP_DATE,'dd-mm-yyyy hh24:mi:ss') OP_DATE , BY_USER
from EMP_SALARY_AUDIT
order by OP_DATE;
```

Trigger Firing Sequence in Oracle

1. **Before Statement**
2. **Before Each Row**
3. **After Each Row**
4. **After Statement**

```
Create table test_emp
(
  emp_id number,
```

```
  First_name varchar2(100)
);
```

```
Create table test_emp_sequence
(
   seq number,
   Trigger_type varchar2(100)
);

create sequence s;


Create or replace trigger before_insert_stat
Before insert
On test_emp
Begin
Insert into test_emp_sequence values ( s.nextval,’before_insert_stat’);
End;

Create or replace trigger before_insert_each_row
Before insert
On test_emp
For each row
Begin
Insert into test_emp_sequence values ( s.nextval,’before_insert_each_row’);
End;

Create or replace trigger after_insert_each_row
after insert
On test_emp
For each row
Begin
Insert into test_emp_sequence values ( s.nextval,’after_insert_each_row’);
End;


Create or replace trigger after_insert_stat
after insert
On test_emp
For each row
Begin
Insert into test_emp_sequence values ( s.nextval,’after_insert_stat’);
End;




insert into test_emp values (1,’abc’);
select * from test_emp_sequence order by seq;

Insert into test_emp
Select employee_id, first_name from employees
Where department_id=20;

select * from test_emp_sequence order by seq;
```

Trigger Options in Oracle

1. **Before Statement**
2. **Before Each Row**
3. **After Each Row**
4. **After Statement**

`alter trigger after_insert_each_row compile;`

`Alter table test_emp disable all triggers;`

`alter table test_emp enable all triggers;`

`alter trigger after_insert_stat disable;`

`alter trigger after_insert_stat enable;`

`drop trigger after_insert_stat;`

```
Create table customers
(
  cust_id number,
  Name varchar2(100),
  Status char(1)
);
```

```
Create or replace trigger customer_default_status
Before insert
On customers
For each row
Begin
:new.status='A';
End;

Insert into customers (cust_id, name) values (1,'abc');
select * from customers;
```