



# Database Programming

## Triggers – Part I

Sagara Samarawickrama  
2023W - CPCM

# Triggers

A database trigger is a named PL/SQL block that is stored in a database and executed implicitly when a triggering event occurs. The act of executing a trigger is referred to as firing the trigger. A triggering event can be any of the following:

**A DML** (for example, INSERT, UPDATE, or DELETE) statement executed against a database table. Such trigger can fire before or after a triggering event. For example, if you have defined a trigger to fire before an INSERT statement on the STUDENT table, this trigger fires each time before you insert a row in the STUDENT table.

**A DDL** (for example, CREATE or ALTER) statement executed either by a particular user against a schema or by any user. Such triggers are often used for auditing purposes and are specifically helpful to Oracle database administrators. They can record various schema changes, including when those changes were made and by which user.

**A system event** such as startup or shutdown of the database.

# Triggers

## *General Syntax for Creating a Trigger*

```
CREATE [OR REPLACE] [EDITIONABLE|NONEDITIONABLE] TRIGGER trigger_name
{BEFORE|AFTER} triggering_event ON table_name
[FOR EACH ROW]
[FOLLOWS|PRECEDES another_trigger]
[ENABLE/DISABLE]
[WHEN condition]
DECLARE
    Declaration statements
BEGIN
    Executable statements
EXCEPTION
    Exception-handling statements
END;
```

The optional reserved words EDITIONABLE and NONEDITIONABLE specify whether a trigger is an editioned or noneditioned object. Note that this designation applies only if editioning has been enabled for object type TRIGGER



# Triggers

The **trigger\_name** references the name of the trigger. **BEFORE** or **AFTER** specifies when the trigger fires (before or after the triggering event). The **triggering\_event** references a DML statement issued against the table. The **table\_name** is the name of the table associated with the trigger. The clause **FOR EACH ROW** specifies that a trigger is a row-level trigger and fires once for each row either inserted, updated, or deleted.

A **WHEN** clause specifies a condition that must evaluate to TRUE for the trigger to fire. For example, this condition may specify a certain restriction on the column of a table.

The next two options, **FOLLOWS/PRECEDES** and **ENABLE/DISABLE**, were added to the CREATE OR REPLACE TRIGGER clause in Oracle 11g

To disable the trigger, you need to issue the ALTER TRIGGER command,

```
ALTER TRIGGER trigger_name DISABLE;
```

# Triggers

Similarly, to enable a trigger that was disabled previously, you issue the ALTER TRIGGER command

```
ALTER TRIGGER trigger_name ENABLE;
```

The **FOLLOWS/PRECEDES** option allows you to specify the order in which triggers should fire. It applies to triggers that are defined on the same table and fire at the same timing point.

The portion of the trigger described to this point is often referred to as the trigger header. Next, we define the trigger body. The body of a trigger has the same structure as an anonymous PL/SQL block. Similar to the case for a PL/SQL block, the declaration and exception sections are optional.



# Triggers

Triggers are used for different purposes, such as the following:

- Triggers are used for different purposes, such as the following:
- Enforcing complex business rules that cannot be defined by using integrity
- constraints
- Maintaining complex security rules
- Automatically generating values for derived columns
- Collecting statistical information on table accesses
- Preventing invalid transactions
- Providing value auditing

# Triggers

## BEFORE Triggers

Consider the following example of a trigger on the STUDENT table mentioned earlier . This trigger fires before the INSERT statement on the STUDENT table and populates the STUDENT\_ID, CREATED\_DATE, MODIFIED\_DATE, CREATED\_BY, and MODIFIED\_BY columns

```
CREATE OR REPLACE TRIGGER student_bi
BEFORE INSERT ON STUDENT
FOR EACH ROW
BEGIN
    :NEW.student_id      := STUDENT_ID_SEQ.NEXTVAL;
    :NEW.created_by      := USER;
    :NEW.created_date    := SYSDATE;
    :NEW.modified_by     := USER;
    :NEW.modified_date   := SYSDATE;
END;
```

This trigger fires for each row before the INSERT statement on the STUDENT table



# Triggers

In the body of the trigger, there is a **pseudorecord**, :NEW, which allows for accessing a row that is currently being processed. In other words, a row is inserted into the STUDENT table.

```
CREATE OR REPLACE TRIGGER student_bi
...
DECLARE
    v_student_id STUDENT.STUDENT_ID%TYPE;
BEGIN
    SELECT STUDENT_ID_SEQ.NEXTVAL
        INTO v_student_id
        FROM dual;

...
END;
```

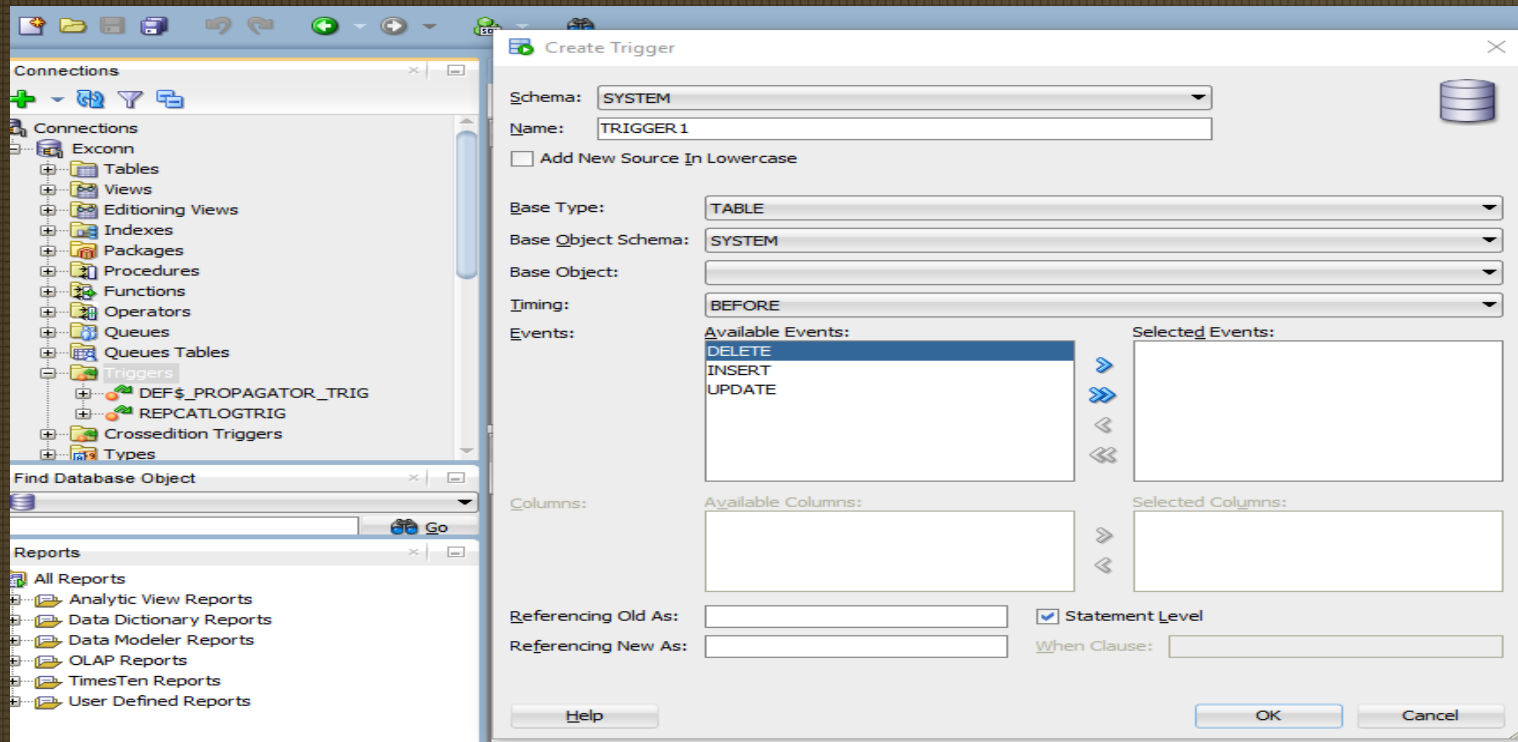
To create this trigger on the STUDENT table in SQL Developer, you may choose from the two options. **First, the trigger can be created by executing the script in the Worksheet window**, just as you would with any other PL/SQL block. At the time of trigger compilation, you are prompted to enter the value for bind variables because of the references to the :NEW and :OLD pseudorecords in the body of the trigger



# Triggers

The second option for creating a trigger is to right-click on Triggers and choose the New Trigger option

This activates the Create Trigger window, as shown in following. In this window, you provide schema name, trigger name, table name, the timing of the triggering event, and the event on which the trigger should fire.



# Triggers

Now that you have created a trigger on the STUDENT table, consider the following INSERT statement.

```
INSERT INTO STUDENT
(student_id, first_name, last_name, zip, registration_date,
created_by, created_date, modified_by, modified_date)
VALUES
(STUDENT_ID_SEQ.NEXTVAL, 'John', 'Smith', '00914', SYSDATE,
USER, SYSDATE, USER, SYSDATE);
```

This INSERT statement contains values for the columns STUDENT\_ID, CREATED\_BY, CREATED\_DATE, MODIFIED\_BY, and MODIFIED\_DATE. Note that for every row you insert into the STUDENT table, you must provide the values for these columns; they are always derived in the same fashion. Now that you have created the trigger, however, there is no need to provide values for these columns in the INSERT statement because the trigger automatically populates these columns in a consistent manner every time an INSERT statement is executed against the STUDENT table.



# Triggers

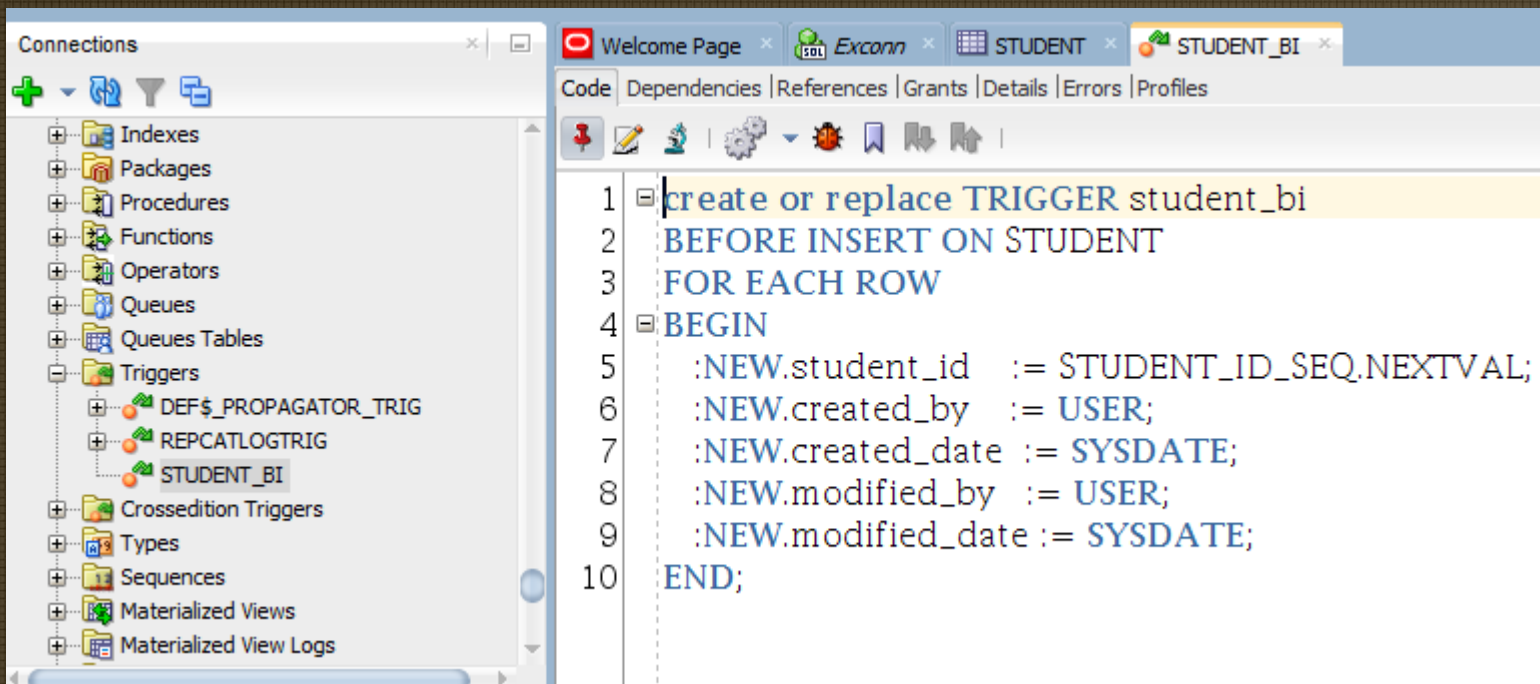
Therefore, the INSERT statement can be modified as follows:

```
INSERT INTO STUDENT  
  (first_name, last_name, zip, registration_date)  
VALUES  
  ('John', 'Smith', '00914', SYSDATE);
```

This version of the INSERT statement is significantly shorter than the previous version. Specifically, instead of providing values for nine columns, you need to provide values for only four columns. The columns **STUDENT\_ID**, **CREATED\_BY**, **CREATED\_DATE**, **MODIFIED\_BY**, and **MODIFIED\_DATE** are no longer present.

# Triggers

Now we will see a working example:  
First create a trigger in SQL Developer



The screenshot shows the SQL Developer interface. On the left, the 'Connections' pane displays a tree view of database objects, including 'Indexes', 'Packages', 'Procedures', 'Functions', 'Operators', 'Queues', 'Queues Tables', 'Triggers', 'Crossedition Triggers', 'Types', 'Sequences', 'Materialized Views', and 'Materialized View Logs'. The 'Triggers' folder is expanded, showing 'DEF\$\_PROPAGATOR\_TRIG', 'REPCATLOGTRIG', and 'STUDENT\_BI'. The main editor window is titled 'STUDENT\_BI' and shows the following SQL code:

```
1 create or replace TRIGGER student_bi
2 BEFORE INSERT ON STUDENT
3 FOR EACH ROW
4 BEGIN
5     :NEW.student_id := STUDENT_ID_SEQ.NEXTVAL;
6     :NEW.created_by  := USER;
7     :NEW.created_date := SYSDATE;
8     :NEW.modified_by := USER;
9     :NEW.modified_date := SYSDATE;
10 END;
```

It will create a before insert trigger and it will fill/update the above mentioned columns automatically



# Triggers

Structure of the table is as follows

	⚡ COLUMN_NAME	⚡ DATA_TYPE	⚡ NULLABLE
1	STUDENT_ID	NUMBER(8,0)	No
2	SALUTATION	VARCHAR2(5 BYTE)	Yes
3	FIRST_NAME	VARCHAR2(25 BYTE)	Yes
4	LAST_NAME	VARCHAR2(25 BYTE)	No
5	STREET_ADDRESS	VARCHAR2(50 BYTE)	Yes
6	ZIP	VARCHAR2(5 BYTE)	No
7	PHONE	VARCHAR2(15 BYTE)	Yes
8	EMPLOYER	VARCHAR2(50 BYTE)	Yes
9	REGISTRATION_DATE	DATE	No
10	CREATED_BY	VARCHAR2(30 BYTE)	No
11	CREATED_DATE	DATE	No
12	MODIFIED_BY	VARCHAR2(30 BYTE)	No
13	MODIFIED_DATE	DATE	No

Since we have created a **before insert trigger** we need not to provide all the values given in this tables we only have to provide values that are not included in the trigger. The columns that I marked in yellow color are included in the trigger.

# Triggers

Now we can write a shorter insert statement as follows

```
INSERT INTO student (SALUTATION,FIRST_NAME,LAST_NAME,STREET_ADDRESS,ZIP,  
PHONE,EMPLOYER,REGISTRATION_DATE)  
VALUES ('Mr.','Sagara','Samarawickrama','81-Pinary Trail.','12345',  
'647-647-6666','Lambton College.',SYSDATE);
```

This will insert the full record to the table. After inserting the record ..

283	397 Ms.	Margaret	Lloyd	77-15 113th Street, #15	11375	718-555-5555	Health & Hospitals
284	399 Mr.	Jerry	Abdou	460 15th St. #4	10025	718-555-5555	Health Mgmt.Systems
285	404 Mr.	Sagara	Samarawickrama	81-Pinary Trail.	12345	647-647-6666	Lambton College.

Remaining columns were added by the trigger

spitals	23-FEB-03	BROSENZWEIG	23-FEB-07	BROSENZW	26-FEB-07
t.Systems	23-FEB-03	BROSENZWEIG	23-FEB-07	BROSENZW	26-FEB-07
llege.	02-JUL-18	SYSTEM	02-JUL-18	SYSTEM	02-JUL-18



