Oracle Procedures

A procedure is a group of PL/SQL statements that can be called by name. The call specification (sometimes called call spec) specifies a java method or a third-generation language routine so that it can be called from SQL and PL/SQL.

Create Procedure

**Syntax**

1. **CREATE** [OR REPLACE] **PROCEDURE** procedure\_name
2. [ (parameter [,parameter]) ]
3. **IS**
4. [declaration\_section]
5. **BEGIN**
6. executable\_section
7. [EXCEPTION
8. exception\_section]
9. **END** [procedure\_name];

Following are the three types of procedures that must be defined to create a procedure.

* **IN:**It is a default parameter. It passes the value to the subprogram.
* **OUT:** It must be specified. It returns a value to the caller.
* **IN OUT:**It must be specified. It passes an initial value to the subprogram and returns an updated value to the caller.

Oracle Create procedure example

In this example, we are going to insert record in the "user" table. So you need to create user table first.

**Table creation:**

1. **create** **table** user(id number(10) **primary** **key**,**name** varchar2(100));

Now write the procedure code to insert record in user table.

**Procedure Code:**

1. **create** or replace **procedure** "INSERTUSER"
2. (id IN NUMBER,
3. **name** IN VARCHAR2)
4. **is**
5. **begin**
6. **insert** **into** user **values**(id,**name**);
7. **end**;
8. /

Output:

Procedure created.

Oracle program to call procedure

Let's see the code to call above created procedure.

1. **BEGIN**
2. insertuser(101,'Rahul');
3. dbms\_output.put\_line('record inserted successfully');
4. **END**;
5. /

Now, see the "USER" table, you will see one record is inserted.

|  |  |
| --- | --- |
| **ID** | **Name** |
| 101 | Rahul |

Oracle Drop Procedure

**Syntax**

1. **DROP** **PROCEDURE** procedure\_name;

Example to drop procedure

1. **DROP** **PROCEDURE** pro1;

Oracle Trigger

In Oracle, you can define procedures that are implicitly executed when an INSERT, UPDATE or DELETE statement is issued against the associated table. These procedures are called database triggers.

There are six CREATE TRIGGER statements according to their firing points.

Firing Point: BEFORE

* BEFORE INSERT TRIGGER
* BEFORE UPDATE TRIGGER
* BEFORE DELETE TRIGGER

Firing Point: AFTER

* AFTER INSERT TRIGGER
* AFTER UPDATE TRIGGER
* AFTER DELETE TRIGGER

Trigger Topics

1) [Oracle BEFORE INSERT/UPDATE/DELETE Trigger](https://www.javatpoint.com/oracle-before-trigger)

2) [Oracle AFTER INSERT/UPDATE/DELETE Trigger](https://www.javatpoint.com/oracle-after-trigger)

3) [Oracle DROP Trigger](https://www.javatpoint.com/oracle-drop-trigger)

4) [Oracle DISABLE Trigger](https://www.javatpoint.com/oracle-disable-trigger)

5) [Oracle ENABLE Trigger](https://www.javatpoint.com/oracle-enable-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.

**Syntax**

1. **CREATE** [ OR REPLACE ] **TRIGGER** trigger\_name
2. BEFORE **INSERT** or **UPDATE** or **DELETE**
3. **ON** table\_name
4. [ **FOR** EACH ROW ]
5. **DECLARE**
6. -- variable declarations
7. **BEGIN**
8. -- trigger code
9. EXCEPTION
10. **WHEN** ...
11. -- exception handling
12. **END**;

Parameters

**OR REPLACE:** It is an optional parameter. It is used to re-create the trigger if it already exists. It facilitates you to change the trigger definition without using a DROP TRIGGER statement.

**trigger\_name:** It specifies the name of the trigger that you want to create.

**BEFORE INSERT or UPDATE or DELETE:** It specifies that the trigger will be fired before the INSERT or UPDATE or DELETE operation is executed.

**table\_name:** It specifies the name of the table on which trigger operation is being performed.

Limitations

* BEFORE trigger cannot be created on a view.
* You cannot update the OLD values.
* You can only update the NEW values.

Oracle BEFORE Trigger Example

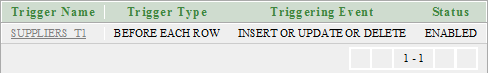
Consider, you have a "suppliers" table with the following parameters.

1. **CREATE** **TABLE**  "SUPPLIERS"
2. (    "SUPPLIER\_ID" NUMBER,
3. "SUPPLIER\_NAME" VARCHAR2(4000),
4. "SUPPLIER\_ADDRESS" VARCHAR2(4000)
5. )
6. /

You can use the following **CREATE TRIGGER query** to create a BEFORE INSERT or UPDATE or DELETE Trigger:

1. **CREATE** OR REPLACE **TRIGGER**  "SUPPLIERS\_T1"
2. BEFORE
3. **insert** or **update** or **delete** **on** "SUPPLIERS"
4. **for** each row
5. **begin**
6. **when** the person performs **insert**/**update**/**delete** operations **into** the **table**.
7. **end**;
8. /
9. **ALTER** **TRIGGER**  "SUPPLIERS\_T1" ENABLE
10. /

Here the trigger name is "SUPPLIERS\_T1" and it is fired BEFORE the insert or update or delete operation is executed on the table "suppliers".



Oracle After INSERT/UPDATE/DELETE Trigger

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

**Syntax**

1. **CREATE** [ OR REPLACE ] **TRIGGER** trigger\_name
2. **AFTER** **INSERT** or **UPDATE** or **DELETE**
3. **ON** table\_name
4. [ **FOR** EACH ROW ]
5. **DECLARE**
6. -- variable declarations
7. **BEGIN**
8. -- trigger code
9. EXCEPTION
10. **WHEN** ...
11. -- exception handling
12. **END**;

Parameters

**OR REPLACE:** It is an optional parameter. It is used to re-create the trigger if it already exists. It facilitates you to change the trigger definition without using a DROP TRIGGER statement.

**trigger\_name:** It specifies the name of the trigger that you want to create.

**AFTER INSERT or UPDATE or DELETE:** It specifies that the trigger will be fired after the INSERT or UPDATE or DELETE operation is executed.

**table\_name:** It specifies the name of the table on which trigger operation is being performed.

Limitations

* AFTER trigger cannot be created on a view.
* You cannot update the OLD values.
* You can only update the NEW values.

Oracle AFTER Trigger Example

Consider, you have a "suppliers" table with the following parameters.

1. **CREATE** **TABLE**  "SUPPLIERS"
2. (    "SUPPLIER\_ID" NUMBER,
3. "SUPPLIER\_NAME" VARCHAR2(4000),
4. "SUPPLIER\_ADDRESS" VARCHAR2(4000)
5. )
6. /

You can use the following **CREATE TRIGGER query** to create a AFTER INSERT or UPDATE or DELETE Trigger:

1. **CREATE** OR REPLACE **TRIGGER**  "SUPPLIERS\_T2"
2. **AFTER**
3. **insert** or **update** or **delete** **on** "SUPPLIERS"
4. **for** each row
5. **begin**
6. **when** the person performs **insert**/**update**/**delete** operations **into** the **table**.
7. **end**;
8. /
9. **ALTER** **TRIGGER**  "SUPPLIERS\_T2" ENABLE
10. /

Here the trigger name is "SUPPLIERS\_T2" and it is fired AFTER the insert or update or delete operation is executed on the table "suppliers".



[**next →**](https://www.javatpoint.com/oracle-disable-trigger)[**← prev**](https://www.javatpoint.com/oracle-after-trigger)

Oracle DROP Trigger

In Oracle, DROP TRIGGER statement is used to drop the trigger if you find that you need to remove it from the database.

**Syntax**

1. **DROP** **TRIGGER** trigger\_name;

Parameters

**trigger\_name:** It specifies the name of the trigger that you want to drop.

Oracle DROP Trigger Example

1. **DROP** **TRIGGER** SUPPLIERS\_T1;

It will drop the trigger name "SUPPLIERS\_T1" from the table "SUPPLIERS".

Oracle DISABLE Trigger

The ALTER TRIGGER statement is used to disable a trigger.

**Syntax**

1. **ALTER** **TRIGGER** trigger\_name DISABLE;

Parameters

**trigger\_name:** It specifies the name of the trigger that you want to disable.

Oracle DISABLE Trigger Example

1. **ALTER** **TRIGGER** SUPPLIERS\_T2 DISABLE;

This example will disable the trigger called "SUPPLIERS\_T2" from the table "SUPPLIERS".

Oracle DISABLE ALL Triggers Example

If there is more than one trigger in a table and you want to disable all the triggers from the database then you can do it by ALTER TABLE statement.

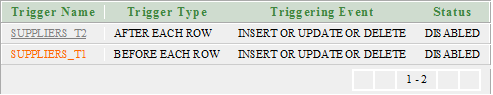
**Syntax**

1. **ALTER** **TABLE** table\_name DISABLE ALL TRIGGERS;

**Example**

1. **ALTER** **TABLE** SUPPLIERS DISABLE ALL TRIGGERS;

This example will disable all triggers from the table "suppliers".



Oracle ENABLE Trigger

The ALTER TRIGGER statement is used to enable a trigger.

**Syntax**

1. **ALTER** **TRIGGER** trigger\_name ENABLE;

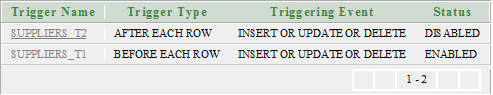
Parameters

**trigger\_name:** It specifies the name of the trigger that you want to enable.

Oracle ENABLE Trigger Example

1. **ALTER** **TRIGGER** SUPPLIERS\_T1 ENABLE;

This example will enable the trigger named "SUPPLIERS\_T1" in the "SUPPLIERS" table.



Oracle ENABLE ALL Triggers Example

**Syntax**

1. **ALTER** **TABLE** table\_name ENABLE ALL TRIGGERS;

**Example**

1. **ALTER** **TABLE** SUPPLIERS ENABLE ALL TRIGGERS;

This example will enable all the triggers on the table name "SUPPLIERS".

