Oracle/PLSQL: BEFORE INSERT Trigger

This Oracle tutorial explains how to **create a BEFORE INSERT Trigger** in Oracle with syntax and examples.

Description

A **BEFORE INSERT Trigger** means that Oracle will fire this trigger before the INSERT operation is executed.

Syntax

The syntax to create a **BEFORE INSERT Trigger** in Oracle/PLSQL is:

```
CREATE [ OR REPLACE ] TRIGGER trigger_name
BEFORE INSERT
   ON table_name
   [ FOR EACH ROW ]

DECLARE
   -- variable declarations

BEGIN
   -- trigger code

EXCEPTION
   WHEN ...
   -- exception handling

END;
```

Parameters or Arguments

OR REPLACE
: Optional. If specified, it allows you to re-create the trigger is it already exists so that you can change the trigger definition without issuing a DROP TRIGGER statement.

trigger_name
: The name of the trigger to create.

BEFORE INSERT
: It indicates that the trigger will fire before the INSERT operation is executed.

table_name
: The name of the table that the trigger is created on.

Restrictions:

- You can not create a BEFORE trigger on a view.
- You can update the :NEW values.
- You can not update the :OLD values.

Note:

- See also how to create AFTER DELETE, AFTER INSERT, AFTER UPDATE, BEFORE DELETE, and BEFORE UPDATE triggers.
- See also how to drop a trigger.

Example

Let's look at an example of how to create an BEFORE INSERT trigger using the CREATE TRIGGER statement.

If you had a table created as follows:

```
CREATE TABLE orders
( order_id number(5),
  quantity number(4),
  cost_per_item number(6,2),
  total_cost number(8,2),
  create_date date,
  created_by varchar2(10)
);
```

We could then use the CREATE TRIGGER statement to create an BEFORE INSERT trigger as follows:

```
CREATE OR REPLACE TRIGGER orders_before_insert
BEFORE INSERT
   ON orders
   FOR EACH ROW

DECLARE
   v_username varchar2(10);

BEGIN

   -- Find username of person performing INSERT into table
   SELECT user INTO v_username
   FROM dual;

   -- Update create_date field to current system date
   :new.create_date := sysdate;

   -- Update created_by field to the username of the person performing
the INSERT
   :new.created_by := v_username;

END;
```

Oracle/PLSQL: AFTER INSERT Trigger

This Oracle tutorial explains how to **create an AFTER INSERT Trigger** in Oracle with syntax and examples.

Description

An **AFTER INSERT Trigger** means that Oracle will fire this trigger after the INSERT operation is executed.

Syntax

The syntax to create an **AFTER INSERT Trigger** in Oracle/PLSQL is:

```
CREATE [ OR REPLACE ] TRIGGER trigger_name
AFTER INSERT
   ON table_name
   [ FOR EACH ROW ]

DECLARE
   -- variable declarations

BEGIN
   -- trigger code

EXCEPTION
   WHEN ...
   -- exception handling

END;
```

Parameters or Arguments

OR REPLACE
: Optional. If specified, it allows you to re-create the trigger is it already exists so that you can change the trigger definition without issuing a DROP TRIGGER statement.

trigger_name
: The name of the trigger to create.

AFTER INSERT
: It indicates that the trigger will fire after the INSERT operation is executed.

table_name
: The name of the table that the trigger is created on.

Restrictions:

- You can not create an AFTER trigger on a view.

- You can not update the :NEW values.
- You can not update the :OLD values.

Example

Let's look at an example of how to create an AFTER INSERT trigger using the CREATE TRIGGER statement.

If you had a table created as follows:

```
CREATE TABLE orders
( order_id number(5),
  quantity number(4),
  cost_per_item number(6,2),
  total_cost number(8,2)
);
```

We could then use the CREATE TRIGGER statement to create an AFTER INSERT trigger as follows:

```
CREATE OR REPLACE TRIGGER orders_after_insert
AFTER INSERT
   ON orders
   FOR EACH ROW

DECLARE
   v_username varchar2(10);

BEGIN

   -- Find username of person performing the INSERT into the table
   SELECT user INTO v_username
   FROM dual;

   -- Insert record into audit table
   INSERT INTO orders_audit
   ( order_id,
     quantity,
     cost_per_item,
     total_cost,
     username )
   VALUES
   ( :new.order_id,
     :new.quantity,
     :new.cost_per_item,
     :new.total_cost,
     v_username );

END;
```

Oracle/PLSQL: BEFORE UPDATE Trigger

This Oracle tutorial explains how to **create a BEFORE UPDATE Trigger** in Oracle with syntax and examples.

Description

A **BEFORE UPDATE Trigger** means that Oracle will fire this trigger before the UPDATE operation is executed.

Syntax

The syntax to create a **BEFORE UPDATE Trigger** in Oracle/PLSQL is:

```
CREATE [ OR REPLACE ] TRIGGER trigger_name
BEFORE UPDATE
   ON table_name
   [ FOR EACH ROW ]

DECLARE
   -- variable declarations

BEGIN
   -- trigger code

EXCEPTION
   WHEN ...
   -- exception handling

END;
```

Parameters or Arguments

OR REPLACE
: Optional. If specified, it allows you to re-create the trigger is it already exists so that you can change the trigger definition without issuing a DROP TRIGGER statement.

trigger_name
: The name of the trigger to create.

BEFORE UPDATE
: It indicates that the trigger will fire before the UPDATE operation is executed.

table_name
: The name of the table that the trigger is created on.

Restrictions:

- You can not create a BEFORE trigger on a view.
- You can update the :NEW values.
- You can not update the :OLD values.

Note:

- See also how to create AFTER DELETE, AFTER INSERT, AFTER UPDATE, BEFORE DELETE, and BEFORE INSERT triggers.
- See also how to drop a trigger.

Example

Let's look at an example of how to create an BEFORE UPDATE trigger using the CREATE TRIGGER statement.

If you had a table created as follows:

```
CREATE TABLE orders
( order_id number(5),
  quantity number(4),
  cost_per_item number(6,2),
  total_cost number(8,2),
  updated_date date,
  updated_by varchar2(10)
);
```

We could then use the CREATE TRIGGER statement to create an BEFORE UPDATE trigger as follows:

```
CREATE OR REPLACE TRIGGER orders_before_update
BEFORE UPDATE
   ON orders
   FOR EACH ROW

DECLARE
   v_username varchar2(10);

BEGIN

   -- Find username of person performing UPDATE on the table
   SELECT user INTO v_username
   FROM dual;

   -- Update updated_date field to current system date
   :new.updated_date := sysdate;

   -- Update updated_by field to the username of the person performing
the UPDATE
   :new.updated_by := v_username;

END;
```

Oracle/PLSQL: AFTER UPDATE Trigger

This Oracle tutorial explains how to **create an AFTER UPDATE Trigger** in Oracle with syntax and examples.

Description

An **AFTER UPDATE Trigger** means that Oracle will fire this trigger after the UPDATE operation is executed.

Syntax

The syntax to create an **AFTER UPDATE Trigger** in Oracle/PLSQL is:

```
CREATE [ OR REPLACE ] TRIGGER trigger_name
AFTER UPDATE
   ON table_name
   [ FOR EACH ROW ]

DECLARE
   -- variable declarations

BEGIN
   -- trigger code

EXCEPTION
   WHEN ...
   -- exception handling

END;
```

Parameters or Arguments

OR REPLACE
: Optional. If specified, it allows you to re-create the trigger is it already exists so that you can change the trigger definition without issuing a DROP TRIGGER statement.

trigger_name
: The name of the trigger to create.

AFTER UPDATE
: It indicates that the trigger will fire after the UPDATE operation is executed.

table_name
: The name of the table that the trigger is created on.

Restrictions:

- You can not create an AFTER trigger on a view.
- You can not update the :NEW values.
- You can not update the :OLD values.

Example

Let's look at an example of how to create an AFTER UPDATE trigger using the CREATE TRIGGER statement.

If you had a table created as follows:

```
CREATE TABLE orders
( order_id number(5),
  quantity number(4),
  cost_per_item number(6,2),
  total_cost number(8,2)
);
```

We could then use the CREATE TRIGGER statement to create an AFTER UPDATE trigger as follows:

```
CREATE OR REPLACE TRIGGER orders_after_update
AFTER UPDATE
   ON orders
   FOR EACH ROW

DECLARE
   v_username varchar2(10);

BEGIN

   -- Find username of person performing UPDATE into table
   SELECT user INTO v_username
   FROM dual;

   -- Insert record into audit table
   INSERT INTO orders_audit
   ( order_id,
     quantity_before,
     quantity_after,
     username )
   VALUES
   ( :new.order_id,
     :old.quantity,
     :new.quantity,
     v_username );

END;
```

Oracle/PLSQL: BEFORE DELETE Trigger

This Oracle tutorial explains how to **create a BEFORE DELETE Trigger** in Oracle with syntax and examples.

Description

A **BEFORE DELETE Trigger** means that Oracle will fire this trigger before the DELETE operation is executed.

Syntax

The syntax to create a **BEFORE DELETE Trigger** in Oracle/PLSQL is:

```
CREATE [ OR REPLACE ] TRIGGER trigger_name
BEFORE DELETE
   ON table_name
   [ FOR EACH ROW ]

DECLARE
   -- variable declarations

BEGIN
   -- trigger code

EXCEPTION
   WHEN ...
   -- exception handling

END;
```

Parameters or Arguments

OR REPLACE
: Optional. If specified, it allows you to re-create the trigger is it already exists so that you can change the trigger definition without issuing a DROP TRIGGER statement.

trigger_name
: The name of the trigger to create.

BEFORE DELETE
: It indicates that the trigger will fire before the DELETE operation is executed.

table_name
: The name of the table that the trigger is created on.

Restrictions:

- You can not create a BEFORE trigger on a view.
- You can update the :NEW values.
- You can not update the :OLD values.

Example

Let's look at an example of how to create an BEFORE DELETE trigger using the CREATE TRIGGER statement.

If you had a table created as follows:

```
CREATE TABLE orders
( order_id number(5),
  quantity number(4),
  cost_per_item number(6,2),
  total_cost number(8,2)
);
```

We could then use the CREATE TRIGGER statement to create an BEFORE DELETE trigger as follows:

```
CREATE OR REPLACE TRIGGER orders_before_delete
BEFORE DELETE
   ON orders
   FOR EACH ROW

DECLARE
   v_username varchar2(10);

BEGIN

   -- Find username of person performing the DELETE on the table
   SELECT user INTO v_username
   FROM dual;

   -- Insert record into audit table
   INSERT INTO orders_audit
   ( order_id,
     quantity,
     cost_per_item,
     total_cost,
     delete_date,
     deleted_by )
   VALUES
   ( :old.order_id,
     :old.quantity,
     :old.cost_per_item,
     :old.total_cost,
      sysdate,
      v_username );

END;
```

Oracle/PLSQL: AFTER DELETE Trigger

This Oracle tutorial explains how to **create an AFTER DELETE Trigger** in Oracle with syntax and examples.

Description

An **AFTER DELETE Trigger** means that Oracle will fire this trigger after the DELETE operation is executed.

Syntax

The syntax to create an **AFTER DELETE Trigger** in Oracle/PLSQL is:

```
CREATE [ OR REPLACE ] TRIGGER trigger_name
AFTER DELETE
   ON table_name
   [ FOR EACH ROW ]

DECLARE
   -- variable declarations

BEGIN
   -- trigger code

EXCEPTION
   WHEN ...
   -- exception handling

END;
```

Parameters or Arguments

OR REPLACE
: Optional. If specified, it allows you to re-create the trigger is it already exists so that you can change the trigger definition without issuing a DROP TRIGGER statement.

trigger_name
: The name of the trigger to create.

AFTER DELETE
: It indicates that the trigger will fire after the DELETE operation is executed.

table_name
: The name of the table that the trigger is created on.

Restrictions:

- You can not create an AFTER trigger on a view.
- You can not update the :NEW values.
- You can not update the :OLD values.

Example

Let's look at an example of how to create an AFTER DELETE trigger using the CREATE TRIGGER statement.

If you had a table created as follows:

```
CREATE TABLE orders
( order_id number(5),
  quantity number(4),
  cost_per_item number(6,2),
  total_cost number(8,2)
);
```

We could then use the CREATE TRIGGER statement to create an AFTER DELETE trigger as follows:

```
CREATE OR REPLACE TRIGGER orders_after_delete
AFTER DELETE
   ON orders
   FOR EACH ROW

DECLARE
   v_username varchar2(10);

BEGIN

   -- Find username of person performing the DELETE on the table
   SELECT user INTO v_username
   FROM dual;

   -- Insert record into audit table
   INSERT INTO orders_audit
   ( order_id,
     quantity,
     cost_per_item,
     total_cost,
     delete_date,
     deleted_by)
   VALUES
   ( :old.order_id,
     :old.quantity,
     :old.cost_per_item,
     :old.total_cost,
     sysdate,
     v_username );

END;
```

Oracle/PLSQL: DROP TRIGGER Statement

This Oracle tutorial explains how to use the **DROP TRIGGER statement** to drop a trigger in Oracle with syntax and examples.

Description

Once you have created a trigger in Oracle, you might find that you need to remove it from the database. You can do this with the DROP TRIGGER statement.

Syntax

The syntax to a **drop a trigger** in Oracle in Oracle/PLSQL is:

```
DROP TRIGGER trigger_name;
```

Parameters or Arguments

trigger_name

The name of the trigger that you wish to drop.

Note: See also how to create AFTER DELETE, AFTER INSERT, AFTER UPDATE, BEFORE DELETE, BEFORE INSERT, and BEFORE UPDATE triggers.

Example

Let's look at an example of how to drop a trigger in Oracle.

For example:

```
DROP TRIGGER orders_before_insert;
```

This example uses the ALTER TRIGGER statement to drop the trigger called *orders_before_insert*.

Oracle/PLSQL: Disable a Trigger

This Oracle tutorial explains how to **disable a trigger** in Oracle with syntax and examples.

Description

Once you have created a Trigger in Oracle, you might find that you are required to disable the trigger. You can do this with the ALTER TRIGGER statement.

Syntax

The syntax for a disabling a Trigger in Oracle/PLSQL is:

```
ALTER TRIGGER trigger_name DISABLE;
```

Parameters or Arguments

trigger_name
: The name of the trigger that you wish to disable.

Note:

- See also how to disable all triggers on a table.
- See also how to enable a trigger on a table or enable all triggers on a table.

Example

Let's look at an example that shows how to disable a trigger in Oracle.

For example:

```
ALTER TRIGGER orders_before_insert DISABLE;
```

This example uses the ALTER TRIGGER statement to disable the trigger called *orders_before_insert*.

Oracle/PLSQL: Disable all Triggers on a table

This Oracle tutorial explains how to **disable all triggers on a table** in Oracle with syntax and examples.

Description

Once you have created Triggers in Oracle, you might find that you are required to disable all of the triggers on a table.. You can do this with the ALTER TRIGGER statement.

Syntax

The syntax for a disabling all Triggers on a table in Oracle/PLSQL is:

```
ALTER TABLE table_name DISABLE ALL TRIGGERS;
```

Parameters or Arguments

table_name
: The name of the table that all triggers should be disabled on.

Note:

- See also how to disable a trigger.
- See also how to enable a trigger on a table or enable all triggers on a table.

Example

Let's look at an example that shows how to disable all triggers on a table in Oracle.

For example:

```
ALTER TABLE orders DISABLE ALL TRIGGERS;
```

This example uses the ALTER TRIGGER statement to disable all triggers on the table called *orders*.

Oracle/PLSQL: Enable a Trigger

This Oracle tutorial explains how to **enable a trigger** in Oracle with syntax and examples.

Description

You may have found that you have disabled a trigger on a table and you wish to enable the trigger again. You can do this with the ALTER TRIGGER statement.

Syntax

The syntax for a enabling a Trigger in Oracle/PLSQL is:

```
ALTER TRIGGER trigger_name ENABLE;
```

Parameters or Arguments

trigger_name
: The name of the trigger that you wish to enable.

Note:

- See also how to enable all triggers on a table.
- See also how to disable a trigger on a table or disable all triggers on a table.

Example

Let's look at an example that shows how to enable a trigger in Oracle.

For example:

```
ALTER TRIGGER orders_before_insert ENABLE;
```

This example uses the ALTER TRIGGER statement to enable the trigger called *orders_before_insert*.

Oracle/PLSQL: Enable all Triggers on a table

This Oracle tutorial explains how to **enable all triggers on a table** in Oracle with syntax and examples.

Description

You may have found that you have disabled all triggers on a table and you wish to enable the triggers again. You can do this with the ALTER TRIGGER statement.

Syntax

The syntax to **enable all triggers on a table** in Oracle/PLSQL is:

```
ALTER TABLE table_name ENABLE ALL TRIGGERS;
```

Parameters or Arguments

table_name
: The name of the table that all triggers should be enabled on.

Note:

- See also how to enable a trigger.
- See also how to disable a trigger on a table or disable all triggers on a table.

Example

Let's look at an example that shows how to enable all triggers on a table in Oracle.

For example:

```
ALTER TABLE orders ENABLE ALL TRIGGERS;
```

This example uses the ALTER TRIGGER statement to enable all triggers on the table called *orders*.