Oracle8 SQL Reference
Release 8.0
A58225-01

CREATE TRIGGER

Purpose

To create and enable a database trigger. A *database trigger* is a stored PL/SQL block associated with a table. Oracle automatically executes a trigger when a specified SQL statement is issued against the table.

Prerequisites

To issue this statement, you must have one of the following system privileges:

CREATE TRIGGER	lets you create a trigger in your own schema on a table in your own schema.
CREATE ANY TRIGGER	lets you create a trigger in any user's schema on a table in any schema.

If the trigger issues SQL statements or calls procedures or functions, then the owner of the schema to contain the trigger must have the privileges necessary to perform these operations. These privileges must be granted directly to the owner, rather than acquired through roles.

Syntax

Keywords and Parameters

OR REPLACE	re-creates the trigger if it already exists. Use this option to change the definition of an existing trigger without first dropping it.
schema	is the schema to contain the trigger. If you omit schema, Oracle creates the trigger in your own schema.
trigger	is the name of the trigger to be created.
BEFORE	causes Oracle to fire the trigger before executing the triggering statement. For row triggers, this is a separate firing before each affected row is changed.
	You cannot specify a BEFORE trigger on a view or an object view.
AFTER	causes Oracle to fire the trigger after executing the triggering statement. For row triggers, this is a separate firing after each affected row is changed.
	You cannot specify an AFTER trigger on a view or an object view.
INSTEAD OF	causes Oracle to fire the trigger instead of executing the triggering statement. By default, INSTEAD OF triggers are activated for each row.
	INSTEAD OF is a valid option only for views. You cannot specify an INSTEAD OF trigger on a table.
DELETE	causes Oracle to fire the trigger whenever a DELETE statement removes a row from the table.
INSERT	causes Oracle to fire the trigger whenever an INSERT statement adds a row to table.

UPDATE	causes Oracle to fire the trigger whenever an UPDATE statement changes a value in one of the columns specified in the OF clause. If you omit the OF clause, Oracle fires the trigger whenever an UPDATE statement changes a value in any column of the table.
	You cannot specify an OF clause with an INSTEAD OF trigger. Oracle fires INSTEAD OF triggers whenever an UPDATE changes a value in any column of the view.

ON	specifies the *schema* and *table* or *view* name of the of one of the following on which the trigger is to be created: • table • object table • view • object view If you omit *schema*, Oracle assumes the table is in your own schema. You can create triggers on index-organized tables. You cannot create a trigger on a table in the schema SYS.
table	is the name of a table or an object table.
view	is the name of a view or an object view.
REFERENCING	specifies correlation names. You can use correlation names in the PL/SQL block and WHEN clause of a row trigger to refer specifically to old and new values of the current row. The default correlation names are OLD and NEW. If your row trigger is associated with a table named OLD or NEW, use this clause to specify different correlation names to avoid confusion between the table name and the correlation name.
FOR EACH ROW	designates the trigger to be a row trigger. Oracle fires a row trigger once for each row that is affected by the triggering statement and meets the optional trigger constraint defined in the WHEN clause. Except for INSTEAD OF triggers, if you omit this clause, the trigger is a statement trigger. Oracle fires a statement trigger only once when the triggering statement is issued if the

	optional trigger constraint is met. INSTEAD OF trigger statements are implicitly activated for each row.

WHEN (*condition*)	specifies the trigger restriction--a SQL condition that must be satisfied for Oracle to fire the trigger. This condition must contain correlation names and cannot contain a query. You can specify a trigger restriction only for a row trigger. Oracle evaluates this condition for each row affected by the triggering statement. You cannot specify trigger restrictions for INSTEAD OF trigger statements. You can reference object columns or their attributes, VARRAY, nested table, or LOB columns. You cannot invoke PL/SQL functions or methods in the trigger restriction.
pl/sql_block	is the PL/SQL block that Oracle executes to fire the trigger.
	Note: The PL/SQL block of a trigger cannot contain transaction control SQL statements (COMMIT, ROLLBACK, SAVEPOINT, and SET CONSTRAINT).

Using Triggers

Oracle automatically *fires*, or executes, a trigger when a triggering statement is issued. You can use triggers for the following purposes:

- to provide sophisticated auditing and transparent event logging
- to automatically generate derived column values
- to enforce complex security authorizations and business constraints
- to maintain replicate asynchronous tables
- etc

An existing trigger must be in one of the following states:

- If a trigger is *enabled*, Oracle fires the trigger whenever a triggering statement is issued and the condition of the trigger restriction is met.
- If a trigger is *disabled*, Oracle does not fire the trigger when a triggering statement is issued and the condition of the trigger restriction is met.

When you create a trigger, Oracle enables it automatically. You can subsequently disable and enable a trigger with the DISABLE and ENABLE options of the ALTER TRIGGER command or the ALTER TABLE command.

```
Alter trigger xxxx disable

Alter table yyyyy disable all trigger
```

Before Release 7.3, Oracle parsed and compiled a trigger whenever it was fired. From Release 7.3 onward, Oracle stores a compiled version of a trigger in the data dictionary and calls this compiled

version when the trigger is fired. This feature provides a significant performance improvement for applications that use many triggers.

If a trigger produces compilation errors, it is still created, but it fails on execution. This means it effectively blocks all triggering DML statements until it is disabled, replaced by a version without compilation errors, or dropped.

Conditional Predicates

When you create a trigger for more than one DML operation, you can use conditional predicates within the trigger body to execute specific blocks of code, depending on the type of statement that fires the trigger. Conditional predicates are evaluated as follows:

INSERTING	returns true if the trigger fires for an INSERT statement.
DELETING	returns true if the trigger fires for a DELETE statement.
UPDATING	returns true if the trigger fires for an UPDATE statement.
UPDATING (*column_name*)	returns true if the trigger fires for an UPDATE statement and *column_name* is updated.

Example

The following example uses conditional predicates to provide information about which DML statement fires trigger AUDIT_TRIGGER:

```
CREATE TRIGGER audit_trigger BEFORE INSERT OR DELETE OR
UPDATE
  ON classified_table FOR EACH ROW
  BEGIN
    IF INSERTING THEN
       INSERT INTO audit_table
          VALUES (USER || ' is inserting' ||
                    ' new key: ' || :new.key);
    ELSIF DELETING THEN
       INSERT INTO audit_table
          VALUES (USER || ' is deleting' ||
                    ' old key: ' || :old.key);
    ELSIF UPDATING('FORMULA') THEN
       INSERT INTO audit_table
          VALUES (USER || ' is updating' ||
                    ' old formula: ' || :old.formula ||
                    ' new formula: ' || :new.formula);
    ELSIF UPDATING THEN
       INSERT INTO audit_table
          VALUES (USER || ' is updating' ||
                    ' old key: ' || :old.key ||
                    ' new key: ' || :new.key);
    END IF;
  END;
```

Parts of a Trigger

The syntax of the CREATE TRIGGER statement includes the following parts of the trigger:

Triggering statement

The definition of the triggering statement specifies what SQL statements cause Oracle to fire the trigger.

DELETE INSERT UPDATE	You must specify at least one of these commands that causes Oracle to fire the trigger. You can specify as many as three.
ON	You must also specify the table with which the trigger is associated. The triggering statement is one that modifies this table. You can define a trigger on an index-organized table.

Trigger restriction

The trigger restriction specifies an additional condition that must be satisfied for a row trigger to be fired. You specify this condition with the WHEN clause. This condition must be a SQL condition, rather than a PL/SQL condition.

Trigger action

The trigger action specifies the PL/SQL block Oracle executes to fire the trigger.

Oracle evaluates the condition of the trigger restriction whenever a triggering statement is issued. If this condition is satisfied, then Oracle fires the trigger using the trigger action.

Types of Triggers

You can create different types of triggers. The type of a trigger determines:

- when Oracle fires the trigger in relation to executing the triggering statement
- how many times Oracle fires the trigger

The type of a trigger depends on the BEFORE, AFTER, and FOR EACH ROW options of the CREATE TRIGGER command. Using all combinations of these options for the above parts, you can create four types of triggers. Table 4-9 describes each type of trigger, its properties, and the options used to create it.

Table 4-9 Types of Triggers

	FOR EACH option	
	STATEMENT	**ROW**
BEFORE Option	**BEFORE statement trigger:** Oracle fires the trigger once before executing the triggering statement.	**BEFORE row trigger:** Oracle fires the trigger before modifying each row affected by the triggering statement.
AFTER Option	**AFTER statement trigger:** Oracle fires the trigger once after executing the triggering statement.	**AFTER row trigger:** Oracle fires the trigger after modifying each row affected by the triggering statement.

For a single table, you can create each type of trigger for each of the following commands:

- DELETE
- INSERT
- UPDATE

You can also create triggers that fire for more than one command.

If you create multiple triggers of the same type that fire for the same command on the same table, the order in which Oracle fires these triggers is indeterminate. If your application requires that one trigger be fired before another of the same type for the same command, combine these triggers into a single trigger whose trigger action performs the trigger actions of the original triggers in the appropriate order.

Example I

This example creates a BEFORE statement trigger named EMP_PERMIT_CHANGES in the schema SCOTT. This trigger ensures that changes to employee records are made only during business hours on working days:

```
CREATE TRIGGER scott.emp_permit_changes
    BEFORE
    DELETE OR INSERT OR UPDATE
    ON scott.emp
    DECLARE
        dummy  INTEGER;
    BEGIN
        /* If today is a Saturday or Sunday,
           then return an error.*/
        IF (TO_CHAR(SYSDATE, 'DY') = 'SAT' OR
         TO_CHAR(SYSDATE, 'DY') = 'SUN')
         THEN raise_application_error( -20501,
       'May not change employee table during the weekend');
        END IF;
       /* Compare today's date with the dates of all
          company holidays. If today is a company holiday,
           then return an error.*/
        SELECT COUNT(*)
         INTO dummy
          FROM company_holidays
          WHERE day = TRUNC(SYSDATE);
        IF dummy > 0
         THEN raise_application_error( -20501,
          'May not change employee table during a holiday');
        END IF;
        /*If the current time is before 8:00AM or after
          6:00PM, then return an error.
        */
        IF (TO_CHAR(SYSDATE, 'HH24') < 8 OR
          TO_CHAR(SYSDATE, 'HH24') >= 18)
          THEN raise_application_error( -20502,
      'May only change employee table during working
hours');
        END IF;
      END;
```

Oracle fires this trigger whenever a DELETE, INSERT, or UPDATE statement affects the EMP table in the schema SCOTT. The trigger EMP_PERMIT_CHANGES is a BEFORE statement trigger, so Oracle fires it once before executing the triggering statement.

The trigger performs the following operations:

1. If the current day is a Saturday or Sunday, the trigger raises an application error with a message that the employee table cannot be changed during weekends.

2. The trigger compares the current date with the dates listed in the table of company holidays.

3. If the current date is a company holiday, the trigger raises an application error with a message that the employee table cannot be changed during holidays.

4. If the current time is not between 8:00AM and 6:00PM, the trigger raises an application error with a message that the employee table can be changed only during business hours.

Example II

This example creates a BEFORE row trigger named SALARY_CHECK in the schema SCOTT. Whenever a new employee is added to the employee table or an existing employee's salary or job is changed, this trigger guarantees that the employee's salary falls within the established salary range for the employee's job:

```
CREATE TRIGGER scott.salary_check
    BEFORE
    INSERT OR UPDATE OF sal, job ON scott.emp
    FOR EACH ROW
    WHEN (new.job <> 'PRESIDENT')
    DECLARE
       minsal NUMBER;
       maxsal NUMBER;
    BEGIN
        /* Get the minimum and maximum salaries for the
          employee's job from the SAL_GUIDE table.  */
        SELECT minsal, maxsal
         INTO minsal, maxsal
         FROM sal_guide
         WHERE job = :new.job;
         /* If the employee's salary is below the minimum or
*/
         /* above the maximum for the job, then generate an
*/
         /* error.*/
        IF (:new.sal < minsal OR :new.sal > maxsal)
        THEN raise_application_error( -20601,
          'Salary ' || :new.sal || ' out of range for job '
          || :new.job || ' for employee ' || :new.ename );
        END IF;
    END;
```

Oracle fires this trigger whenever one of the following statements is issued:

- an INSERT statement that adds rows to the EMP table
- an UPDATE statement that changes values of the SAL or JOB columns of the EMP table

SALARY_CHECK is a BEFORE row trigger, so Oracle fires it before changing each row that is updated by the UPDATE statement or before adding each row that is inserted by the INSERT statement.

SALARY_CHECK has a trigger restriction that prevents it from checking the salary of the company president. For each new or modified employee row that meets this condition, the trigger performs the following steps:

1. The trigger queries the salary guide table for the minimum and maximum salaries for the employee's job.

2. The trigger compares the employee's salary with these minimum and maximum values.

3. If the employee's salary does not fall within the acceptable range, the trigger raises an application error with a message that the employee's salary is not within the established range for the employee's job.

INSTEAD OF Triggers

Use INSTEAD OF triggers to perform DELETE, UPDATE, or INSERT operations on views, which are not inherently modifiable.

In the following example, customer data is stored in two tables. The object view ALL_CUSTOMERS is created as a UNION of the two tables, CUSTOMERS_SJ and CUSTOMERS_PA. An INSTEAD OF trigger is used to insert values:

```
CREATE TABLE customers_sj
  ( cust    NUMBER(6),
    address VARCHAR2(50),
    credit   NUMBER(9,2)  );


CREATE TABLE customers_pa
  ( cust    NUMBER(6),
    address VARCHAR2(50),
    credit   NUMBER(9,2) );


CREATE TYPE customer_t AS OBJECT
  ( cust    NUMBER(6),
    address   VARCHAR2(50),
    credit    NUMBER(9,2),
    location   VARCHAR2(20)  );


CREATE VIEW all_customers (cust)
AS SELECT customer_t (cust, address, credit, 'SAN_JOSE')
FROM   customers_sj
UNION ALL
SELECT customer_t(cust, address, credit, 'PALO_ALTO')
FROM   customers_pa;
```

```
CREATE TRIGGER instrig INSTEAD OF INSERT ON all_customers
   FOR EACH ROW
      BEGIN
        IF (:new.location = 'SAN_JOSE') THEN
           INSERT INTO customers_sj
            VALUES (:new.cust, :new.address, :new.credit);
        ELSE
           INSERT INTO customers_pa
            VALUES (:new.cust, :new.address, :new.credit);
        END IF;
       END;
```