8

Triggers

What is a Trigger

- A PL/SQL block
- Stored program
- Associated with a specific action (an event) on a database object
- Fires automatically when the event occurs

Comparing Triggers to Procedures & Functions

- Figure 8-2 provides a comparison between triggers and procedures.

Triggers	Procedures
Defined with `CREATE TRIGGER`	Defined with `CREATE PROCEDURE`
Cannot receive or return parameters	Can receive and return parameters
Implicitly invoked by the database	Explicitly invoked in a block
Data Dictionary contains source code in `USER_TRIGGERS`	Data Dictionary contains source code in `USER_SOURCE`
`COMMIT`, `SAVEPOINT`, and `ROLLBACK` are not allowed	`COMMIT`, `SAVEPOINT`, and `ROLLBACK` are allowed

- Figure 8-2

Benefits of Triggers

- Enforcing complex business rules that cannot be established using integrity constraint such as `UNIQUE`, `NOT NULL`, and `CHECK`.
- Creating logging records
- Imposing security authorizations
- Preventing invalid transactions
- Auditing sensitive data

Elements of Triggers

- The elements of a trigger are level, timing, and event as identified in Figure 8-1.

Level	`STATEMENT`	Fires once for the entire statement
	`ROW`	Fires once for each row affected by the triggering event (only DML events)
Timing	`BEFORE`	Fires before the specified event occurs
	`AFTER`	Fires after the specified event occurs
	`INSTEAD OF`	A special type of trigger for DML events.
Event	DML Trigger	Fires on a database manipulation (DML) statement (`DELETE`, `INSERT`, or `UPDATE`).
	DDL Trigger	Fires on a database definition (DDL) statement (`CREATE`, `ALTER`, or `DROP`).
	Database Trigger	Fires on a database operation (`LOGON`, `LOGOFF`, `STARTUP`, `SHUTDOWN`, or `SERVERERROR`).

Figure 8-1

- The result of executing a trigger is also known as firing a trigger. We say that the trigger fired.

Initial Setup for Examples

- The procedure `INITIAL_SETUP` shown in Figure 8-3 is used to initialize the environment for the statement and row trigger examples
- This procedure is executed before each example.
- Before each example, the environment is initialized or reset using the anonymous block in Figure 8-4.

```
BEGIN
  initial_setup;
END;
```

Figure 8-4

Statement Trigger

- The STATEMENT level trigger is the default for the trigger level
- Figure 8-5 shows the syntax for a statement level trigger
- To specify a ROW level trigger, FOR EACH ROW is specified
- Row level triggers are discussed later in this chapter.

- A trigger is broken into two components: header and body
- Figure 8-6 shows the trigger header.

```
CREATE [OR REPLACE] TRIGGER trigger_name
    [BEFORE | AFTER]
    [INSERT | [OR] UPDATE [OF column_name(s)] | [OR] DELETE]
    ON table_name
    [FOR EACH ROW]
    [WHEN condition]
```

Figure 8-6

- Figure 8-7 shows the trigger body which has the same structure as an anonymous block.

```
DECLARE
  -- declaration section
BEGIN
  -- execution section
EXCEPTION
  -- exception-handling section
END;
```

Figure 8-7

- The DECLARE, BEGIN, and EXCEPTION sections are the same as other PL/SQL blocks. The DECLARE and EXCEPTION sections are optional

A Statement Trigger:

- Fires only once whenever the triggering event occurs (even if no rows are affected)
- Is the default type of trigger
- Useful if the trigger body does not need to process column values from affected rows

Timing

- `BEFORE` triggers are often used to decide whether the triggering DML statement should be allowed to complete
- `AFTER` triggers are frequently used to generate audit records and log records
- If a trigger throws an unhandled error, the trigger, plus its triggering SQL statement would be rolled back. This would happen regardless of whether a `BEFORE` or `AFTER` trigger is used

Statement Trigger Example 1: BEFORE Trigger

- Automatically fires `BEFORE` an `UPDATE` or `DELETE` operation is performed
- An `INSERT` statement performed on the `c8_employees` table will not fire the trigger
- Regardless of how many rows are updated or deleted, the statement trigger fires only once.

```
CREATE OR REPLACE TRIGGER stmt_1_trg
BEFORE UPDATE OR DELETE ON c8_employees
BEGIN
  INSERT INTO c8_log_stmt_updates (trg_name, trg_event)
         VALUES ('stmt_1_trg', 'BEFORE-UPDATE OR DELETE');
  DBMS_OUTPUT.PUT_LINE ('stmt_1_trg BEFORE UPDATE OR DELETE completed.');
END;
```

```
Trigger created.
```

Figure 8-8

- The `UPDATE` statement in Figure 8-9 invokes the `stmt_1_trg` trigger.

```
UPDATE c8_employees      -- one row updated
  SET salary = 4100
  WHERE emp_id = 1;
```

```
stmt_1_trg BEFORE UPDATE OR DELETE completed.
1 row(s) updated.
```

Figure 8-9

- The `UPDATE` statement in Figure 8-10 invokes the `stmt_1_trg` trigger.

```
UPDATE c8_employees      -- one row updated
  SET dept_id = 30
  WHERE emp_id = 2;
```

```
stmt_1_trg BEFORE UPDATE OR DELETE completed.
1 row(s) updated.
```

Figure 8-10

- In Figure 8-11, the employees in department 20 are given a 5% salary increase
- There are 5 employees in department 20
- How many rows are updated in the table?
- How many times did the trigger fire?

```sql
UPDATE c8_employees
  SET salary = salary * 1.05
  WHERE dept_id = 20;
```

```
stmt_1_trg BEFORE UPDATE OR DELETE completed.
5 row(s) updated.
```

Figure 8-11

- In Figure 8-12, all employees in department 40 are deleted.

```sql
DELETE FROM c8_employees
WHERE dept_id = 40;
```

```
stmt_1_trg BEFORE UPDATE OR DELETE completed.
3 row(s) deleted.
```

Figure 8-12

- In Figure 8-13, employee 0 does not exists
- No rows are updated in the employees table
- How many times did the trigger fire?

```sql
UPDATE c8_employees
  SET salary = 9900
  WHERE emp_id = 0;
```

```
stmt_1_trg BEFORE UPDATE OR DELETE completed.
0 row(s) updated.
```

Figure 8-13

- In Figure 8-14, the update is attempting to give employees in department 90 a 5% salary increase
- There are no employees in department 90
- How many rows are updated?
- How many times did the trigger fire?

```
UPDATE c8_employees
  SET salary = salary * 1.05
  WHERE dept_id = 90;
```

```
stmt_1_trg BEFORE UPDATE OR DELETE completed.
0 row(s) updated.
```

Figure 8-14

- The log table is examined in Figure 8-15 to verify the correct number of entries
- There is one entry in the table for every time the trigger is fired.

```
SELECT *
FROM c8_log_stmt_updates
ORDER BY log_id;
```

```
LOG_ID   TRG_NAME     TRG_EVENT

     1   stmt_1_trg   BEFORE-UPDATE OR DELETE
     2   stmt_1_trg   BEFORE-UPDATE OR DELETE
     3   stmt_1_trg   BEFORE-UPDATE OR DELETE
     4   stmt_1_trg   BEFORE-UPDATE OR DELETE
     5   stmt_1_trg   BEFORE-UPDATE OR DELETE
     6   stmt_1_trg   BEFORE-UPDATE OR DELETE
```

Figure 8-15

Statement Trigger Example 2: AFTER Trigger

- Automatically fires `AFTER` an `UPDATE` operation is performed
- `INSERT` and `DELETE` operations do not cause the trigger to fire
- Regardless of how many rows are updated, the statement trigger fires only once

```
CREATE OR REPLACE TRIGGER stmt_2_trg
AFTER UPDATE ON c8_employees
BEGIN
  INSERT INTO c8_log_stmt_updates (trg_name, trg_event)
         VALUES ('stmt_2_trg', 'AFTER-UPDATE');
  DBMS_OUTPUT.PUT_LINE ('stmt_2_trg AFTER UPDATE completed.');
END;
```

```
Trigger created.
```

Figure 8-16

- The `UPDATE` statement in Figure 8-17 invokes the `stmt_2_trg` trigger where the salary for employee 3 is updated.

```
UPDATE c8_employees
  SET salary = 4100
  WHERE emp_id = 3;
```

```
stmt_2_trg AFTER UPDATE completed.
1 row(s) updated.
```

Figure 8-17

- In Figure 8-18, all employees in department 30 are given a 2.5% salary increase
- There are 3 employees in department 30
- How many rows are updated in the table?
- How many times did the trigger fire?

```
UPDATE c8_employees
  SET salary = salary * 1.025
  WHERE dept_id = 30;
```

```
stmt_2_trg AFTER UPDATE completed.
3 row(s) updated.
```

Figure 8-18

- One entry in the log table for every time the trigger is fired

```
SELECT *
FROM c8_log_stmt_updates
ORDER BY log_id;
```

```
LOG_ID   TRG_NAME     TRG_EVENT

     1   stmt_2_trg   AFTER-UPDATE
     2   stmt_2_trg   AFTER-UPDATE
```

Figure 8-19

Statement Trigger Example 3: Restricts UPDATE Event to One Column

- Automatically fires BEFORE an UPDATE operation is performed on the salary column
- If a row is inserted or deleted, the trigger will not fire
- If an UPDATE is performed on a column other the salary column, the trigger will not fire

```
CREATE OR REPLACE TRIGGER stmt_3_trg
BEFORE UPDATE OF salary ON c8_employees
BEGIN
  INSERT INTO c8_log_stmt_updates (trg_name, trg_event)
       VALUES ('stmt_3_trg', 'BEFORE-UPDATE');
  DBMS_OUTPUT.PUT_LINE ('stmt_3_trg BEFORE UPDATE completed.');
END;
```

```
Trigger created.
```

Figure 8-20

- In Figure 8-21, salary for employee 4 is updated.

```
UPDATE c8_employees
  SET salary = 4100
  WHERE emp_id = 4;
```

```
stmt_3_trg BEFORE UPDATE completed.
1 row(s) updated.
```

Figure 8-21

- In Figure 8-22, dept_id is updated for employee 4
- The stmt_3_trg trigger is not fired because it only fires when an update is done on salary

```
UPDATE c8_employees
  SET dept_id = 30
  WHERE emp_id = 4;
```

```
1 row(s) updated.
```

Figure 8-22

- In Figure 8-23, there is one entry in the table for every time the statement trigger is fired.

```
SELECT *
FROM c8_log_stmt_updates
ORDER BY log_id;
```

```
LOG_ID   TRG_NAME     TRG_EVENT

     1   stmt_3_trg   BEFORE-UPDATE
```

Figure 8-23

Statement Trigger Example 4: Restrict UPDATE Trigger to More than One Column

- Automatically fires `BEFORE` an `UPDATE` operation is performed on either `salary` or `dept_id`
- If a row is inserted or deleted, the trigger will not fire
- If an `UPDATE` is performed on columns other the `salary` or `dept_id` columns, the trigger will not fire

```
CREATE OR REPLACE TRIGGER stmt_4_trg
BEFORE UPDATE OF salary, dept_id ON c8_employees
BEGIN
  INSERT INTO c8_log_stmt_updates (trg_name, trg_event)
         VALUES ('stmt_4_trg', 'BEFORE-UPDATE');
  DBMS_OUTPUT.PUT_LINE ('stmt_4_trg salary completed.');
END;
```

```
Trigger created.
```

Figure 8-24

- In Figure 8-25, the salary for employee 5 is updated

```
UPDATE c8_employees
  SET salary = 4300  -- UPDATE salary
  WHERE emp_id = 5;
```

```
stmt_4_trg salary completed.
1 row(s) updated.
```

Figure 8-25

- In Figure 8-26, the `dept_id` for employee 5 is changed to 20

```
UPDATE c8_employees
  SET dept_id = 20   -- UPDATE dept_id
  WHERE emp_id = 5;
```

```
stmt_4_trg salary completed.
1 row(s) updated.
```

Figure 8-26

- In Figure 8-27, `salary` is changed to `3900` and `dept_id` is changed to `50` for employee `6`.

```
UPDATE c8_employees
  SET salary = 3900, dept_id = 50  -- UPDATE salary and department id
  WHERE emp_id = 6;
```

```
stmt_4_trg salary completed.
1 row(s) updated.
```

Figure 8-27

- In Figure 8-28, all employees in department `40` are given a 3.5% salary increase
- `dept_id` is changed to `50`.

```
UPDATE c8_employees
  SET salary = salary * 1.035, dept_id = 50  -- UPDATE salary and dept id
  WHERE dept_id = 40;                        -- all employees in dept 40
```

```
stmt_4_trg salary completed.
2 row(s) updated.
```

Figure 8-28

- Log table, one entry in the table for every time the statement trigger is fired

```
SELECT *
FROM c8_log_stmt_updates
ORDER BY log_id;
```

```
LOG_ID   TRG_NAME     TRG_EVENT

     1   stmt_4_trg   BEFORE-UPDATE
     2   stmt_4_trg   BEFORE-UPDATE
     3   stmt_4_trg   BEFORE-UPDATE
     4   stmt_4_trg   BEFORE-UPDATE
```

Figure 8-29

Statement Trigger Example 5: Multiple Triggering Events

- Automatically BEFORE an INSERT, UPDATE, or DELETE operation
- Uses a CASE statement to determine the operation being performed

```
CREATE OR REPLACE TRIGGER stmt_5_trg
BEFORE INSERT OR UPDATE OR DELETE ON c8_employees
DECLARE
  v_trg_event  VARCHAR2(30);
BEGIN
  CASE
    WHEN INSERTING THEN
      v_trg_event := 'BEFORE-INSERT';
    WHEN UPDATING THEN
      v_trg_event := 'BEFORE-UPDATE';
    WHEN DELETING THEN
      v_trg_event := 'BEFORE-DELETE';
  END CASE;
  INSERT INTO c8_log_stmt_updates (trg_name, trg_event)
         VALUES ('stmt_5_trg', v_trg_event);
  DBMS_OUTPUT.PUT_LINE ('Trigger stmt_5_trg completed.');
END;
```

```
Trigger created.
```

Figure 8-30

- In Figure 8-31, a new row is inserted into the `c8_employees` table.

```
INSERT INTO c8_employees VALUES (10, 70, 5000);
```

```
Trigger stmt_5_trg completed.
1 row(s) inserted.
```

Figure 8-31

- In Figure 8-32, the `salary` for employee `7` is updated to `3400`.

```
UPDATE c8_employees
  SET salary = 3400
  WHERE emp_id = 7;
```

```
stmt_5_trg salary completed.
1 row(s) updated.
```

Figure 8-32

- In Figure 8-33, employee `8` is deleted.

```
DELETE
FROM c8_employees
WHERE emp_id = 8;
```

```
Trigger stmt_5_trg completed.
1 row(s) deleted.
```

Figure 8-33

- Log table - one entry in the table for every time the statement trigger is fired.

```
SELECT *
FROM c8_log_stmt_updates
ORDER BY log_id;
```

```
LOG_ID   TRG_NAME     TRG_EVENT

     1   stmt_5_trg   BEFORE-INSERT
     2   stmt_5_trg   BEFORE-UPDATE
     3   stmt_5_trg   BEFORE-DELETE
```

Figure 8-34

Statement Trigger Example 6: Enforce Complex Business Rules

- A statement trigger can be used to enforce complex business rules that cannot be enforced by a constraint
- Automatically fires `BEFORE` an `UPDATE` operation
- `INSERT`s are only allowed during normal working days (Monday through Friday), but prevent `INSERT`s on the weekend (Saturday and Sunday)
- Uses an `IF` statement to check when the update is being performed
- If the day if Saturday or Sunday, the trigger raises an exception error

```
CREATE OR REPLACE TRIGGER stmt_6_trg
BEFORE UPDATE ON c8_employees
BEGIN
  IF TO_CHAR(SYSDATE,'DY') IN ('SAT','SUN') THEN
    RAISE_APPLICATION_ERROR (-20500,'UPDATE during business hours only');
  END IF;
END;
```

```
Trigger created.
```

Figure 8-35

- The `RAISE_APPLICATION_ERROR` is a server-side, built-in procedure that returns an unhandled exception to the user
- Causes the trigger to fail
- When it fails, the triggering statement is automatically rolled back and the employee is not inserted.

Why is this a `BEFORE` trigger?

- If it were an `AFTER` trigger, by the time the trigger raised the error, it would be too late, the employee would already have been inserted into the `c8_employees` table.

Row Triggers

- The `STATEMENT` level trigger is the default for the trigger level
- To specify a `ROW` level trigger, **`FOR EACH ROW`** is specified
- A row trigger fires (executes) once for each row affected by the triggering DML statement

```
CREATE OR REPLACE TRIGGER emp_upd_trg
AFTER UPDATE OF salary ON employees
FOR EACH ROW
BEGIN
  ...
END;
```

Row Trigger Example 0: Using :OLD and :NEW Clauses

- Row triggers fire for each row
- Sometimes a requirement to store the before and after values of the data being changed
- There are two clauses used to store these two values:
  - `:OLD` contains the old or original value of the column before execution of the DML statement
  - `:NEW` contains the new value of the column after execution of the DML statement
- Must be preceded with a colon (`:`)
- `:OLD` and `:NEW` are external variable references
- Two situations where the `:OLD` and `:NEW` clauses do not contain a value:

Triggering Event	:OLD	:NEW
INSERT	No	Yes
UPDATE	Yes	Yes
DELETE	Yes	No

Figure 8-51

Row Trigger Example 00: Using :OLD and :NEW Clauses with INSERT

- Automatically fires `AFTER` each row `INSERT` into the `c8_employees table`.

```
CREATE OR REPLACE TRIGGER row_00_trg
AFTER INSERT ON c8_employees
FOR EACH ROW
BEGIN
  DBMS_OUTPUT.PUT_LINE('Employee: '    || :OLD.emp_id);
  DBMS_OUTPUT.PUT_LINE('Old salary: ' || :OLD.dept_id);
  DBMS_OUTPUT.PUT_LINE('Old salary: ' || :NEW.dept_id);
  DBMS_OUTPUT.PUT_LINE('Old salary: ' || :OLD.salary);
  DBMS_OUTPUT.PUT_LINE('New salary: ' || :NEW.salary);
END;
```

```
Trigger created.
```

Figure 8-54

- What are the values of `:OLD.emp_id`, `:OLD.dept_id`, and `:OLD.salary`?
- The values are `NULL` because the row did not exist before the `INSERT`

```
INSERT INTO c8_employees VALUES (10, 40, 3250);
```

```
Old Employee:
New Employee: 10
Old Dept ID:
New Dept ID: 40
Old salary:
New salary: 3250
```

Figure 8-55

Row Trigger Example 1: BEFORE Trigger

- Automatically fires `BEFORE` each row
- `salary` column is `UPDATE`d
- Will not fire if a column other than the salary column is updated
- Inserts a row into the log table whenever there is an update on the salary column

```
CREATE OR REPLACE TRIGGER row_1_trg
BEFORE UPDATE OF salary ON c8_employees
FOR EACH ROW
BEGIN
  INSERT INTO c8_log_salary_updates (trg_name, emp_id, old_salary, new_salary)
  VALUES ('row_1_trg', :OLD.emp_id, :OLD.salary, :NEW.salary);
  DBMS_OUTPUT.PUT_LINE ('row_1_trg BEFORE UPDATE completed.');
END;
```

```
Trigger created.
```

Figure 8-57

- In Figure 8-58, the salary for employee `1` is changed to `4100`.

```
UPDATE c8_employees      -- one row updated
  SET salary = 4100
  WHERE emp_id = 1;
```

```
row_1_trg BEFORE UPDATE completed.
1 row(s) updated.
```

Figure 8-58

- In Figure 8-59, `dept_id` for employee `6` is changed to `30`
- **Does not fire the trigger because this trigger only fires when salary is updated**

```
UPDATE c8_employees  -- one row updated, but no row inserted into log table
  SET dept_id = 30
  WHERE emp_id = 6;
```

```
1 row(s) updated.
```

Figure 8-59

- In Figure 8-60, employees in department 20 are given a 5% salary increase
- There are 3 employees in department 20
- How many rows are updated in the table?
- How many times did the trigger fire?

```
UPDATE c8_employees
  SET salary = salary * 1.05
  WHERE dept_id = 20;
```

```
row_1_trg BEFORE UPDATE completed.
row_1_trg BEFORE UPDATE completed.
row_1_trg BEFORE UPDATE completed.

3 row(s) updated.
```

Figure 8-60

- In Figure 8-61, employee `0` does not exists
- No rows are updated in the `c8_employees` table
- No rows are inserted into the `c8_log_salary_updates` table

```
UPDATE c8_employees
  SET salary = 9900
  WHERE emp_id = 0;
```

```
0 row(s) updated.
```

Figure 8-61

- In Figure 8-62, attempting to give employees in department 90 a 5% salary increase
- There are no employees in department 90
- No rows are updated in the `c8_employees` table
- No rows are inserted into the `c8_log_salary_updates` table

```
UPDATE c8_employees
  SET salary = salary * 1.05
  WHERE dept_id = 90;
```

```
0 row(s) updated.
```

Figure 8-62

- Log table - There is one entry in the table for every row the trigger is fired

```
SELECT *
FROM c8_log_salary_updates
ORDER BY log_id;
```

```
LOG_ID   TRG_NAME   EMP_ID   OLD_SALARY   NEW_SALARY
     1   row_1_trg       1         3975         4100
     2   row_1_trg       2         3522         3698
     3   row_1_trg       4         3950         4148
     4   row_1_trg       9         3703         3888
```

Figure 8-63

Row Trigger Example 4: Multiple Triggering Events

- Automatically fires `AFTER` an `INSERT`, `UPDATE`, or `DELETE` operation
- Uses a `CASE` statement to determine the operation being performed

```
CREATE OR REPLACE TRIGGER row_4_trg
AFTER INSERT OR UPDATE OR DELETE ON c8_employees
FOR EACH ROW
DECLARE
  v_trg_event    c8_log_row_updates.trg_event%TYPE;
BEGIN
  CASE
    WHEN INSERTING THEN
      v_trg_event := 'BEFORE-INSERT';
    WHEN UPDATING THEN
      v_trg_event := 'BEFORE-UPDATE';
    WHEN DELETING THEN
      v_trg_event := 'BEFORE-DELETE';
  END CASE;
  INSERT INTO c8_log_row_updates (trg_name, trg_event, emp_id, dept_id, salary)
         VALUES ('row_4_trg', v_trg_event, :NEW.emp_id, :NEW.dept_id, :NEW.salary);
  DBMS_OUTPUT.PUT_LINE ('Trigger row_4_trg completed.');
END row_4_trg;
```

```
Trigger created.
```

Figure 8-75

Row Trigger Example 5: UPDATE on Specific Columns

- `IF` conditional statements are used to test for `UPDATE` on specific columns
- The keyword `UPDATING`, and the column names are upper case
- Only fired when the `salary` and `dept_id` columns are updated.

```
CREATE OR REPLACE TRIGGER row_5_trg
AFTER UPDATE OF salary, dept_id ON c8_employees
FOR EACH ROW
BEGIN
  IF UPDATING('SALARY') THEN
    INSERT INTO c8_log_salary_updates(trg_name, emp_id, old_salary, new_salary)
         VALUES ('row_5_trg', :OLD.emp_id, :OLD.salary, :NEW.salary);
  END IF;
  IF UPDATING('DEPT_ID') THEN
    INSERT INTO c8_log_dept_updates
                (trg_name, emp_id, old_dept, new_dept)
         VALUES ('row_5_trg', :OLD.emp_id, :OLD.dept_id, :NEW.dept_id);
  END IF;
  DBMS_OUTPUT.PUT_LINE ('Trigger row_5_trg completed.');
EXCEPTION
   WHEN OTHERS THEN
      raise_application_error (-20002,'Invalid update operation.');
END row_5_trg;
```

```
Trigger created.
```

Figure 8-80

Row Trigger Example 6: Restrict Column

- In Figure 8-86, an `IF` conditional statement is used to provide conditions in which the salary column can be updated

```
CREATE OR REPLACE TRIGGER row_6_trg
BEFORE INSERT OR UPDATE OF salary ON c8_employees
FOR EACH ROW
BEGIN
  IF (:OLD.dept_id IN (20, 40)) AND :NEW.salary > 4000 THEN
     :NEW.salary := 4000;
     INSERT INTO c8_log_salary_limit
                 (trg_name, emp_id, old_dept_id, old_salary, new_salary)
     VALUES ('row_6_trg', :OLD.emp_id, :OLD.dept_id, :OLD.salary, :NEW.salary);
  END IF;
END;
```

```
Trigger created.
```

Figure 8-86