

## Working with Compound Triggers

- The compound trigger body supports a common PL/SQL state that the code for each timing point can access.
- The compound trigger common state is:
  - Established when the triggering statement starts
  - Destroyed when the triggering statement completes
- A compound trigger has a declaration section and a section for each of its timing points.

18-2

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

ORACLE

### What Is a Compound Trigger?

Starting with Oracle Database 11g, you can use a compound trigger. A compound trigger is a single trigger on a table that allows you to specify actions for each of the four triggering timing points:

Before the firing statement

Before each row that the firing statement affects

After each row that the firing statement affects

After the firing statement

Note: For additional information about triggers, refer to the Oracle Database PL/SQL Language Reference 11g Release 2 (11.2).

### Working with Compound Triggers

The compound trigger body supports a common PL/SQL state that the code for each timing point can access. The common state is automatically destroyed when the firing statement completes, even when the firing statement causes an error. Your applications can avoid the mutating table error by allowing rows destined for a second table (such as a history table or an audit table) to accumulate and then bulk-inserting them.

Before Oracle Database 11g Release 1 (11.1), you needed to model the common state with an ancillary package. This approach was both cumbersome to

## The Benefits of Using a Compound Trigger

You can use compound triggers to:

- Program an approach where you want the actions you implement for the various timing points to share common data.
- Accumulate rows destined for a second table so that you can periodically bulk-insert them
- Avoid the mutating-table error (ORA-04091) by allowing rows destined for a second table to accumulate and then bulk-inserting them

ORACLE

18-3

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

program and subject to memory leak when the firing statement caused an error and the after-statement trigger did not fire. Compound triggers make PL/SQL easier for you to use and improve run-time performance and scalability.

## Timing-Point Sections of a Table Compound Trigger

A compound trigger defined on a table has one or more of the following timing-point sections. Timing-point sections must appear in the order shown in the table.

Timing Point	Compound Trigger Section
Before the triggering statement executes	BEFORE statement
After the triggering statement executes	AFTER statement
Before each row that the triggering statement affects	BEFORE EACH ROW
After each row that the triggering statement affects	AFTER EACH ROW

ORACLE

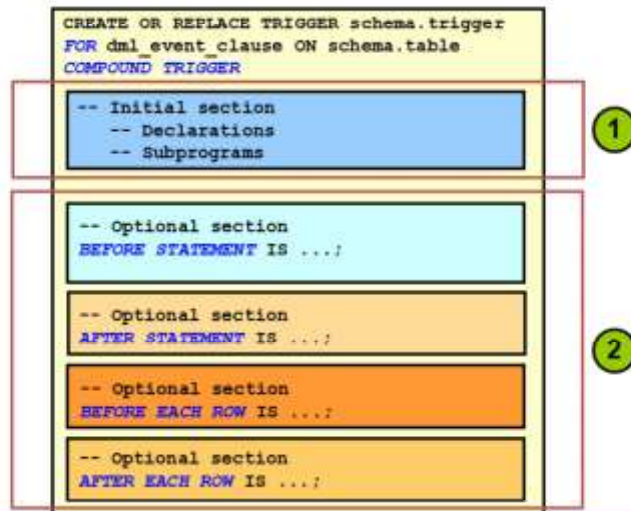
18-4

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Note

Timing-point sections must appear in the order shown in the slide. If a timingpoint section is absent, nothing happens at its timing point.

## Compound Trigger Structure for Tables



ORACLE

18-5

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

### Compound Trigger Structure for Tables

A compound trigger has two main sections:

An initial section where variables and subprograms are declared. The code in this section executes before any of the code in the optional section.

An optional section that defines the code for each possible trigger point. Depending on whether you are defining a compound trigger for a table or for a view, these triggering points are different and are listed in the image shown above and on the following page. The code for the triggering points must follow the order shown above.

Note: For additional information about Compound Triggers, refer to Oracle Database PL/SQL Language Reference 11g Release 2 (11.2).

## Compound Trigger Restrictions

- A compound trigger must be a DML trigger and defined on either a table or a view.
- The body of a compound trigger must be compound trigger block, written in PL/SQL.
- A compound trigger body cannot have an initialization block; therefore, it cannot have an exception section.
- An exception that occurs in one section must be handled in that section. It cannot transfer control to another section.
- :OLD and :NEW cannot appear in the declaration, BEFORE STATEMENT, or the AFTER STATEMENT sections.
- Only the BEFORE EACH ROW section can change the value of :NEW.
- The firing order of compound triggers is not guaranteed unless you use the FOLLOWS clause.

ORACLE

18-7

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

### Compound Trigger Structure for Views

With views, the only allowed section is an INSTEAD OF EACH ROW clause.

### Compound Trigger Restrictions

The following are some of the restrictions when working with compound triggers:

The body of a compound trigger must compound trigger block, written in PL/SQL.

A compound trigger must be a DML trigger.

A compound trigger must be defined on either a table or a view.

A compound trigger body cannot have an initialization block; therefore, it cannot have an exception section. This is not a problem, because the BEFORE STATEMENT section always executes exactly once before any other timing-point section executes.

An exception that occurs in one section must be handled in that section. It cannot transfer control to another section.

:OLD, :NEW, and :PARENT cannot appear in the declaration section, the BEFORE STATEMENT section, or the AFTER STATEMENT section.

## Trigger Restrictions on Mutating Tables

- A mutating table is:
  - A table that is being modified by an `UPDATE`, `DELETE`, or `INSERT` statement, or
  - A table that might be updated by the effects of a `DELETE CASCADE` constraint
- The session that issued the triggering statement cannot query or modify a mutating table.
- This restriction prevents a trigger from seeing an inconsistent set of data.
- This restriction applies to all triggers that use the `FOR EACH ROW` clause.
- Views being modified in the `INSTEAD OF` triggers are not considered mutating.

ORACLE

18-8

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

The firing order of compound triggers is not guaranteed unless you use the `FOLLOWS` clause.

### Rules Governing Triggers

Reading and writing data using triggers is subject to certain rules. The restrictions apply only to row triggers, unless a statement trigger is fired as a result of `ONDELETECASCADE`.

#### Mutating Table

A mutating table is a table that is currently being modified by an `UPDATE`, `DELETE`, or `INSERT` statement, or a table that might need to be updated by the effects of a declarative `DELETECASCADE` referential integrity action. For `STATEMENT` triggers, a table is not considered a mutating table.

A mutating table error (`ORA-4091`) occurs when a row-level trigger attempts to change or examine a table that is already undergoing change via a DML statement.

The triggered table itself is a mutating table, as well as any table referencing it with the `FOREIGNKEY` constraint. This restriction prevents a row trigger from seeing an inconsistent set of data.

## Mutating Table: Example

```
UPDATE employees
SET salary = 3400
WHERE last_name = 'Stiles';
```

```
TRIGGER check_salary Compiled.

Error starting at line 1 in command:
UPDATE employees
SET salary = 3400
WHERE last_name = 'Stiles'
Error report:
ORA-06091: table ORA42.EMPLOYEES is mutating, trigger/function may not see it
ORA-06512: at "ORA42.CHECK_SALARY", line 3
ORA-04088: error during execution of trigger 'ORA42.CHECK_SALARY'
ORA-06091. 00000 - "table %s.%s is mutating, trigger/function may not see it"
*Cause:      A trigger (or a user defined plsql function that is referenced in
             this statement) attempted to look at (or modify) a table that was
             in the middle of being modified by the statement which fired it.
*Action:     Rewrite the trigger (or function) so it does not read that table.
```

ORACLE

18-10

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

### Mutating Table: Example

The CHECK\_SALARY trigger in the slide example attempts to guarantee that whenever a new employee is added to the EMPLOYEES table or whenever an existing employee's salary or job ID is changed, the employee's salary falls within the established salary range for the employee's job.

When an employee record is updated, the CHECK\_SALARY trigger is fired for each row that is updated. The trigger code queries the same table that is being updated. Therefore, it is said that the EMPLOYEES table is a mutating table.

### Mutating Table: Example (continued)

In the example in the slide, the trigger code tries to read or select data from a mutating table.

If you restrict the salary within a range between the minimum existing value and the maximum existing value, then you get a run-time error. The EMPLOYEES table is mutating, or in a state of change; therefore, the trigger cannot read from it.

Remember that functions can also cause a mutating table error when they are invoked in a DML statement.

### Possible Solutions

Possible solutions to this mutating table problem include the following:

Use a compound trigger as described earlier in this lesson. Store the summary data (the minimum salaries and the maximum salaries) in another summary table, which is kept up-to-date with other DML triggers.

Store the summary data in a PL/SQL package, and access the data from the package. This can be done in a BEFORE statement trigger. Depending on the nature of the problem, a solution can become more convoluted and difficult to solve. In this case, consider implementing the rules



in the application or middle tier and avoid using database triggers to perform overly complex business rules. An insert statement in the code example in the slide will not generate a mutating table example.

## Using a Compound Trigger to Resolve the Mutating Table Error

```
CREATE OR REPLACE TRIGGER check_salary
FOR INSERT OR UPDATE OF salary, job_id
ON employees
WHEN (NEW.job_id <> 'AD_PRES')
COMPOUND TRIGGER

TYPE salaries_t          IS TABLE OF employees.salary%TYPE;
min_salaries             salaries_t;
max_salaries             salaries_t;

TYPE department_ids_t    IS TABLE OF employees.department_id%TYPE;
department_ids           department_ids_t;

TYPE department_salaries_t IS TABLE OF employees.salary%TYPE
INDEX BY VARCHAR2(80);
department_min_salaries  department_salaries_t;
department_max_salaries  department_salaries_t;

-- example continues on next slide
```

ORACLE

18-11

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

### Using a Compound Trigger to Resolve the Mutating Table Error

The CHECK\_SALARY compound trigger resolves the mutating table error in the earlier example. This is achieved by storing the values in PL/SQL collections, and then performing a bulk insert/update in the “before statement” section of the compound trigger. This is a 11g only code. In the example in the slide, PL/SQL collections are used. The element types used are based on the SALARY and DEPARTMENT\_ID columns from the EMPLOYEES table. To create collections, you define a collection type, and then declare variables of that type. Collections are instantiated when you enter a block or subprogram, and cease to exist when you exit. min\_salaries is used to hold the minimum salary for each department and max\_salaries is used to hold the maximum salary for each department. department\_ids is used to hold the department IDs. If the employee who earns the minimum or maximum salary does not have an assigned department, you use the NVL function to store -1 for the department id instead of NULL. Next, you collect the minimum salary, maximum salary, and the department ID using a bulk insert into the min\_salaries, max\_salaries, and department\_ids respectively grouped by department ID. The select statement returns 13 rows. The values of the department\_ids are used as an index for the

Oracle Database: Develop PL/SQL

Program Units 9 - 11

department\_min\_salaries and department\_max\_salaries tables. Therefore, the index for those two tables (VARCHAR2) represents the actual department\_ids.

## Using a Compound Trigger to Resolve the Mutating Table Error

```

--
--
--
BEFORE STATEMENT IS
BEGIN
    SELECT MIN(salary), MAX(salary), NVL(department_id, -1)
    BULK COLLECT INTO min_salaries, max_salaries, department_ids
    FROM employees
    GROUP BY department_id;
    FOR j IN 1..department_ids.COUNT() LOOP
        department_min_salaries(department_ids(j)) := min_salaries(j);
        department_max_salaries(department_ids(j)) := max_salaries(j);
    END LOOP;
END BEFORE STATEMENT;

AFTER EACH ROW IS
BEGIN
    IF :NEW.salary < department_min_salaries(:NEW.department_id)
    OR :NEW.salary > department_max_salaries(:NEW.department_id) THEN
        RAISE_APPLICATION_ERROR(-20505, 'New Salary is out of acceptable
        range');
    END IF;
END AFTER EACH ROW;
END check_salary;

```

ORACLE

18-12

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using a Compound Trigger to Resolve the Mutating Table Error (continued) After each row is added, if the new salary is less than the minimum salary for that department or greater than the department's maximum salary, then an error message is displayed.

To test the newly created compound trigger, issue the following statement:

```

UPDATE employees
SET salary = 3400
WHERE last_name = 'Stiles';

```

```

1 rows updated

```

To ensure that the salary for employee Stiles was updated, issue the following

## Oracle Database: Develop PL/SQL

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	JOB_ID	DEPARTMENT_ID	SALARY
1	138	Stephen	Stiles	ST_CLERK	50	3400

Program Units 9 - 12

query using the F9 key in SQL Developer:

```
SELECT employee_id, first_name, last_name, job_id,  
       department_id, salary  
FROM employees  
WHERE last_name = 'Stiles';
```

